Calcolatori Elettronici - Architettura e Organizzazione

Appendice B

Rappresentazione dell'informazione

Giacomo Bucci

Revisione del 31 marzo 2017

Questo documento è una appendice al volume
Calcolatori Elettronici - Architettura e Organizzazione
IV edizione
McGraw-Hill Education (Italy), S.r.l.
Milano, 2017

Storia degli aggiornamenti

Marzo 2017: primo rilascio.

B

Rappresentazione dell'informazione

OBIETTIVI

- Illustrare la numerazione posizionale, esprimere la rappresentazione dei numeri in funzione della base di numerazione
- Mostrare come si effettua la conversione tra le rappresentazioni in basi diverse
- Introdurre l'aritmetica binaria, numeri positivi, numeri negativi, numeri in virgola fissa e in virgola mobile; operazioni in aritmetica binaria
- Mostrare la costruzione di unità aritmetiche e logiche
- Introdurre la rappresentazione dell'informazione di tipo alfanumerico
- Illustrare alcuni standard di rappresentazione in virgola mobile

CONCETTI CHIAVE

Numero, sistema posizionale, base della rappresentazione, conversione della base, aritmetica binaria, rappresentazione in complemento, numeri frazionari, numeri virgola mobile, standard IEEE, informazione non numerica.

INTRODUZIONE

Nei calcolatori elettronici l'elemento primario di informazione è il cosiddetto "bit" (termine derivato da ***b**inary dig**it***), ovvero un'entità che prende valori sull'insieme $\{0,\ 1\}$. All'interno della macchina, l'informazione è rappresentata attraverso opportune tecniche di codifica su raggruppamenti di bit. Anche i numeri sono espressi in forma binaria e le operazioni tra numeri avvengono secondo la corrispondente aritmetica binaria.

Scopo di questo capitolo è illustrare questa aritmetica, in forma intera e in virgola mobile, e mostrare la codifica dell'informazione non numerica.

Sull'aritmetica, intera e in virgola mobile, e sulla rappresentazione dell'informazione, esiste un'ampia letteratura, sia di carattere generale [Bar91], [HP06], [PH07], [HVZ02], sia specialistico [Omo94]alla quale il lettore è invitato a riferirsi per ulteriori approfondimenti.

Il capitolo mostra la costruzione di una unità aritmetica e logica, per estensioni successive a partire da componenti elementari.

B.1 Numerazione posizionale

Gli antichi Greci usavano i numeri in modo, per così dire, operativo, senza porsi il problema di definire "cosa fosse un numero". Il problema della definizione del concetto di numero è stato posto nella seconda metà del diciannovesimo secolo dal matematico e filosofo tedesco Gottlob Frege, iniziatore del cosiddetto *logicismo*. Bertrand Russell, che pure fece parte del programma logicista, spiega che quello di *numero* è un concetto astratto e corrisponde alla descrizione quantitativa degli oggetti contenuti in un dato insieme [Rus70]. Un *sistema di numerazione* è un insieme di simboli e regole atti a rappresentare i numeri.

Il sistema di numerazione usuale è il sistema posizionale[1] decimale. Prendiamo il numero 1475. Esso viene interpretato come:

$$1475 = 1 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 5 \times 10^0$$

ovvero, ogni cifra che compare in 1475 assume un valore che dipende dalla posizione (peso) nella stringa "1475". Il sistema di numerazione decimale si basa su 10 simboli (cifre) diversi $\{0, 1, 2, ..., 9\}$, con i quali si possono rappresentare tutti i possibili numeri, come sequenze di cifre diverse.

La ragione per cui l'uomo conta in base dieci deriva sicuramente dal numero di dita delle nostre mani. Si sa di popolazioni che contavano in base 5. I Maya avevano un sistema di numerazione vigesimale.

In generale, dato un numero $B \geq 2$, detto base, e dato l'insieme β composto da B simboli diversi: $\beta = \{0, 1, 2, ..., B-1\}$, la stringa di n cifre

$$b_{n-1} b_{n-2} \cdots b_1 b_0$$

con $b_i \in \beta$ si interpreta come:

$$b_{n-1} \times B^{n-1} + b_{n-2} \times B^{n-2} + \cdots + b_1 \times B^1 + b_0 \times B^0$$

B.1.1 Esempi di numeri in basi diverse

Vengono ora mostrati alcuni esempi di numeri in basi diverse da 10.

Base 16

Se $B = 16$ siamo nel caso del sistema esadecimale. Per costruire un insieme di 16 simboli diversi si prendono a prestito le prime 6 lettere dell'alfabeto, per cui

$$\beta = \{0, 1, 2, ..., 9, A, B, C, D, E, F\}$$

La stringa 22, interpretata in base 16, corrisponde al numero

$$22 = 2 \times 16^1 + 2 \times 16^0 = 34$$

[1] Gli antichi Romani usavano un sistema di numerazione di carattere additivo. Anche tale sistema faceva uso di un numero ristretto di simboli, ma le regole con cui le varie cifre concorrevano alla quantificazione del numero erano di tipo additivo.

L'apparente incongruenza deriva dal fatto che normalmente, quando si vede un numero scritto, se ne dà un'interpretazione decimale. Per evitare equivoci, l'uguaglianza andrebbe scritta come

$$22_{16} = 34_{10}$$

Il concetto di numero è puramente astratto: 22 e 34 sono la rappresentazione dello stesso numero in due diverse basi di numerazione. Per questo motivo, stringhe di cifre come 22 e 34, di base imprecisata, si dicono *numerali*. Ovviamente, quando l'interpretazione di un numerale non dà luogo a equivoci l'indicazione della base viene omessa.

Base 8
Se $B = 8$ siamo nel caso del sistema ottale. L'insieme dei simboli diversi è $\beta = \{0, 1, 2, ..., 7\}$. La stringa 417 corrisponde al numero:

$$417 = 4 \times 8^2 + 1 \times 8^1 + 7 \times 8^0 = 4 \times 64 + 1 \times 8 + 7 \times 1 = 271$$

Base 3
Se $B = 3$ siamo nel caso del sistema ternario. L'insieme dei simboli diversi è $\beta = \{0, 1, 2\}$. Alla stringa 1021, corrisponde il numero

$$1 \times 3^3 + 0 \times 3^2 + 2 \times 3^1 + 1 \times 2^0 = 1 \times 27 + 0 \times 9 + 2 \times 3 + 1 \times 1 = 34_{10}$$

Base 2
Se $B = 2$ siamo nel caso del sistema binario. L'insieme dei simboli diversi si riduce a $\beta = \{0, 1\}$. La stringa 10011, corrisponde al numero

$$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 19_{10}$$

In Tabella B.1 vengono riportati i primi 17 numeri interi nelle basi 10, 2, 3, 4, 5, 8, 16.

La numerazione in base 2 è importante perché nei calcolatori elettronici l'informazione è rappresentata solo attraverso due simboli $\{0, 1\}$. Le numerazioni in base 16 e in base 8 interessano perché la trasformazione tra queste basi e la base 2 (e viceversa) è immediata. La numerazione in base 8, popolare fino agli anni settanta, è ormai in disuso.

B.2 Conversione di base

B.2.1 Conversione tra base 10 e base 2

Da binario a decimale
La conversione da binario a decimale si effettua come calcolo del polinomio di potenze del 2. Ad esempio, Dato il numero binario 1001101, il corrispondente decimale si ottiene calcolando il polinomio:

$$1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 8 + 4 + 1$$

ovvero $1001101_2 = 77_{10}$

Base 10	Base 2	Base 3	Base 4	Base 5	Base 8	Base 16
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	10	2	2	2	2	2
3	11	10	3	3	3	3
4	100	11	10	4	4	4
5	101	12	11	10	5	5
6	110	20	12	11	6	6
7	111	21	13	12	7	7
8	1000	22	20	13	10	8
9	1001	100	21	14	11	9
10	1010	101	22	20	12	A
11	1011	102	23	21	13	B
12	1100	110	30	22	14	C
13	1101	111	31	23	15	D
14	1110	112	32	24	16	E
15	1111	120	33	30	17	F
16	10000	121	100	31	20	10

Tabella B.1 I primi 17 numeri nelle basi 10, 2, 3, 5, 8 e 16.

Da decimale a binario

La conversione a binario del numero decimale N richiede che si trovi la stringa di n cifre binarie $b_{n-1}b_{n-2}\cdots b_1b_0$ con $b_i = 0, 1$, tale per cui

$$N = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$$

Se si divide il polinomio per 2 si ottiene b_0 come resto e $b_{n-1}\times 2^{n-2}+b_{n-2}\times 2^{n-3}++\cdots+b_1$ come quoziente. Dividendo il quoziente per 2 si ottiene b_1 come resto e $b_{n-1} \times 2^{n-3} + b_{n-2}\times 2^{n-4}+\cdots+b_2$ come quoziente. In altre parole, la ricerca dei coefficienti $b-i$ richiede che si iteri il procedimento fino a che l'ultimo quoziente ottenuto non è più divisibile. A quel punto la rappresentazione binaria si ottiene scrivendo da sinistra verso destra i resti in ordine inverso rispetto a quello secondo cui sono stati prodotti.

Esempio

Dato numero 35_{10}, la serie successiva di quozienti e resti ottenuti dividendo per 2 è:

$$(17, 1), (8, 1), (4, 0), (2, 0), (1, 0)(0, 1)$$

dunque la rappresentazione binaria del numero 35 (decimale) è: 100011.

B.2.2 Conversione tra base B^k e base B

Sia data la stringa: $b_{n-1}b_{n-2}\cdots b_1b_0$ in base B^k, cui corrisponde il numero

$$b_{n-1}(B^k)^{n-1} + b_{n-2}(B^k)^{n-2} + \cdots + b_1(B^k)^1 + b_0(B^k)^0 \tag{B.1}$$

dove ogni b_i è preso da $\{0, 1, ..., B^k - 1\}$ ed è rappresentato in base B come

$$b_{i,k-1}b_{i,k-2}\cdots b_{i,0} = b_{i,k-1}B^{k-1} + b_{i,k-2}B^{k-2} + \cdots + b_{i,0}B^0$$

dove ogni $b_{i,j}$ è preso da $\{0, 1, ..., B - 1\}$. Sostituendo in (B.1) a ciascun b_i la sua rappresentazione in base B, si ha

$$
\begin{aligned}
& [b_{n-1,k-1}B^{k-1} + b_{n-1,k-2}B^{k-2} + \cdots + b_{n-1,0}B^0](B^k)^{n-1} + \\
+ \quad & [b_{n-2,k-1}B^{k-1} + b_{n-2,k-2}B^{k-2} + \cdots + b_{n-2,0}B^0](B^k)^{n-2} + \cdots \\
+ \quad & [b_{0,k-1}B^{k-1} + b_{0,k-2}B^{k-2} + \cdots + b_{0,0}B^0](B^k)^0 = \\
= \quad & b_{n-1,k-1}B^{kn-1} + b_{n-1,k-2}B^{kn-2} + \cdots \\
+ \quad & b_{0,k-1}B^{k-1} + b_{0,k-2}B^{k-2} + \cdots + b_{0,0}B^0
\end{aligned}
$$

che corrisponde alla stringa $b_{n-1,k-1}b_{n-1,k-2}{\cdots}b_{n-1,0}b_{n-2,k-2}{\cdots}b_{0,0}$. Dunque la conversione da base B^k a base B consiste nel reinterpretare la stringa $b_{n-1}b_{n-2}\cdots b_1b_0$ in base B^k, sostituendo a ciascun b_i della base B^k la corrispondente rappresentazione in base B.

Conversione esadecimale-binario

Di particolare interesse è la conversione tra esadecimale e binario. Si consideri, per esempio, il numero esadecimale 1AB07. La sostituzione ordinata dei digit esadecimale è questa:

$$1AB07 = 0001\,1010\,1011\,0000\,0111$$

ovvero $1AB07_{16} = 00011010101100000111_2$

Il processo viene applicato in modo inverso per la conversione da binario a esadecimale. Ad esempio, dato il numero binario 010111000011, si raggruppano a partire da destra le cifre binarie a quattro a quattro (0101 1100 0011) e si sostituisce ciascun gruppo con la corrispondente cifra esadecimale (Tabella B.1). Si ottiene così la stringa 5C3, rappresentazione esadecimale del numero binario di partenza.

B.2.3 Conversione tra generiche basi

In linea teorica è possibile passare direttamente da una base all'altra se si riesce a fare i conti in un generico sistema posizionale. Per evitare questa difficoltà basta passare dalla base di partenza alla base 10 e da questa alla base di arrivo. Se, ad esempio, è dato il numero 143 in base 5 e lo si vuole convertire in base 3, si converte prima 143_5 in base 10 col calcolo del polinomio seguente

$$143_5 = 1 \times 5^2 + 4 \times 5^1 + 3 \times 5^0 = 48_{10}$$

quindi si applica al numero 48_{10} l'algoritmo delle successive divisioni per 3, ottenendo questa sequenza di quozienti e resti:

$$(16, 0), (5, 1), (1, 2), (0, 1)$$

a cui corrisponde il numero 1210_3. Ovvero: $143_5 = 1210_3$.

B.3 Aritmetica binaria

La costruzione di un'aritmetica binaria richiede che vengano scalati sull'insieme $\{0, 1\}$ i familiari concetti dell'aritmetica decimale. In particolare si possono costruire le tabelline delle varie operazioni aritmetiche per la rappresentazione binaria. Le motivazioni per le quali le informazioni all'interno di un calcolatore sono in forma binaria sono illustrate al Paragrafo A.1 dell'Appendice A.

Somma

In Figura B.1 viene riportata la tabellina della somma e un esempio di somma di due numeri. La somma viene eseguita esattamente come nel sistema decimale, partendo da destra verso sinistra, tenendo conto dei riporti.

0 + 0 = 0	
0 + 1 = 1	
1 + 0 = 1	
1 + 1 = 0	*e riporto di* 1

1 1 1 1		*riporti*
1 1 0 0 1 0 1 1	+	*I addendo*
1 1 0 1 1 1 0		*II addendo*
1 0 0 1 1 1 0 0 1		

Figura B.1 Tabellina della somma aritmetica binaria ed esempio di somma tra due numeri (1100 1011 e 110 1110).

Sottrazione

In Figura B.2 viene riportata la tabellina della sottrazione e un esempio di differenza tra due numeri. La sottrazione viene eseguita, come nel sistema decimale, partendo da destra verso sinistra e tenendo conto dei prestiti.

0 − 0 = 0	
0 − 1 = 1	*e prestito di* 1
1 − 0 = 1	
1 − 1 = 0	

1 1 1 1		*prestiti*
1 0 0 1 1 1 0 0 1	−	*minuendo*
1 1 0 0 1 0 1 1		*sottraendo*
0 1 1 0 1 1 1 0		

Figura B.2 Tabellina della sottrazione in aritmetica binaria ed esempio di sottrazione tra due numeri (1 0011 1001 e 1100 1011). Come minuendo è stato preso il risultato della somma di Figura B.1, come sottraendo è il primo addendo della medesima somma. Il risultato dà necessariamente il primo addendo della stessa somma. Si osservi che se si sottrae 1 da 0 il risultato è 1, ma c'è un prestito (dalla cifra più a sinistra).

Moltiplicazione

In Figura B.3 viene riportata la tabellina del prodotto e un esempio di moltiplicazione tra due numeri. La moltiplicazione è un processo che richiede il calcolo dei prodotti parziali e, alla fine, il calcolo della loro somma, esattamente come nell'aritmetica in base dieci.

Divisione

La divisione tra binari (Figura B.4) si effettua in modo del tutto analogo a quella decimale, procedendo per sottrazioni tra parti del dividendo e del divisore. Ovviamente, come nel sistema decimale, la divisione per zero non è definita.

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

```
  1 0 1 1 0 ×
      1 0 1
  ---------
  1 0 1 1 0
0 0 0 0 0
1 0 1 1 0
-----------
1 1 0 1 1 1 0
```

Figura B.3 Tabellina del prodotto in aritmetica binaria ed esempio di moltiplicazione tra due numeri binari interi senza segno (1 0110 e 101). Il prodotto è calcolato col medesimo procedimento seguito in aritmetica decimale.

$0 : 1 = 0$
$1 : 1 = 1$

```
1 1 0 1 0 0 1 1 0 1 : 1 0 0 0 1
1 0 0 0 1             1 1 0 0 0 1
  1 0 0 1 0
  1 0 0 0 1
        1 1 1 0 1
        1 0 0 0 1
          1 1 0 0
```

Figura B.4 Tabellina della divisione in aritmetica binaria ed esempio di divisione tra due numeri (11 0100 1101 e 1 0001). Il primo numero è pari a 845_{10}, il secondo a 17_{10}. La divisione dà 11 0001 (49_{10}) come quoziente e 1100 (12_{10}) come resto.

Al Paragrafo B.9 viene illustrata costruzione di una rete logica in grado di eseguire la somma di due generici numeri binari. Una rete per la sottrazione, ottenuta estendendo la rete per la somma, viene illustrata al Paragrafo B.9.4. Al Paragrafo B.9.6 viene illustrata una rete che svolge il prodotto tra due numeri binari positivi, mentre al Paragrafo B.9.8 viene illustrata una rete per la divisione.

B.4 Numeri negativi

Fino ad ora abbiamo considerato solo numeri positivi, senza badare al numero di cifre, assumendo implicitamente che l'1 più a sinistra fosse il bit più significativo. In un calcolatore i numeri sono rappresentati su gruppi di bit (parole[2]) di dimensione prefissata (ad esempio 8, 16, 32). Preso un vettore di n cifre binarie

$$B = b_{n-1}b_{n-2}.......b_1b_0,$$

con $b_i \in \{0,1\}$, il vettore rappresenta 2^n numeri diversi, per esempio i 2^n interi positivi compresi tra 0 e $2^n - 1$.

Per rappresentare i numeri negativi occorre stabilire una qualche convenzione. Di norma, se il bit più a sinistra della parola è 1, allora il numero viene interpretato come negativo. Ci sono due convenzioni principali.

[2] Usiamo qui il termine generico *parola* per indicare un raggruppamento di bit. Nel caso di parole di 8 bit si usa il termine *byte*.

Rappresentazione in modulo e segno Si passa da valore positivo a negativo semplicemente cambiando da 0 a 1 il bit più significativo.
Rappresentazione in complemento Si passa da valore positivo a negativo effettuando il complemento a 1 o a 2.

La rappresentazione in modulo e segno richiede che la macchina sia equipaggiata con l'unità per eseguire le sottrazioni, mentre, come vedremo, la rappresentazione in complemento richiede solo il circuito di somma. Per questo motivo i numeri interi negativi vengono solitamente rappresentati in complemento.

B.4.1 Rappresentazione in complemento dei numeri binari

Sono possibili il complemento a 1 e il complemento a 2. Con il complemento a 1 il cambiamento di segno viene ottenuto complementando ciascun bit. Con il complemento a 2 il cambiamento di segno viene ottenuto complementando a 1 e aggiungendo 1.

In Tabella B.2 vengono riportati i numeri su 4 bit nella nella notazione in modulo e segno e nelle due differenti notazioni in complemento. Qualunque sia la notazione scelta i numeri positivi vanno da 0 a $2^{n-1} - 1$. I numeri negativi vanno da -0 a $-(2^{n-1} - 1)$ con la notazione in modulo e segno e con la notazione in complemento a 1; ovvero, la notazione in modulo e segno e quella in complemento a 1 hanno doppia rappresentazione dello zero ($+0$ e -0), mentre con la notazione in complemento a 2 i numeri negativi non hanno la doppia rappresentazione dello 0 e vanno da -1 a -2^{n-1}.
Ad esempio, nella notazione in complemento a 2, con un byte (8 bit) i numeri interi positivi vanno da 0 a 127, i negativi da -1 a -128; con una parola di 16 bit il massimo numero intero positivo rappresentabile è 32767, mentre i negativi da vanno da -1 a -32768; con una parola di 32 bit il massimo numero intero positivo rappresentabile è 4.294.967.295 il minimo intero negativo rappresentabile è $-4.294.967.2966$.

Positivi o nulli		Negativi o nulli			
	Tutte le notazioni		Segno e modulo	Complem. a 1	Complem. a 2
+0	0000	-0	1000	1111	
+1	0001	-1	1001	1110	1111
+2	0010	-2	1010	1101	1110
+3	0011	-3	1011	1100	1101
+4	0100	-4	1100	1011	1100
+5	0101	-5	1101	1010	1011
+6	0110	-6	1110	1001	1010
+7	0111	-7	1111	1000	1001
		-8			1000

Tabella B.2 Rappresentazioni binarie di interi su 4 bit. Si noti la doppia rappresentazione dello 0 nelle notazioni in modulo e segno e in complemento a 1.

La soluzione in complemento a 2 è quella normalmente usata, per l'univocità dello zero e per la minor macchinosità del processo di calcolo. Infatti, come illustrato nell'Esempio seguente, la sottrazione del numero b da a viene eseguita come somma di a col complemento a 2 di b.

Esempio

Sottrarre il numero 0001 0110 (pari a 22_{10}) da 0001 1110 (pari a 30_{10}) e da 0001 0011 (pari a 19_{10}); i numeri sono rappresentati su 8 bit.
Il complemento a 2 del sottraendo si ottiene come:

0001 0110	numero dato ($= +22_{10}$)
1110 1001	complemento a 1
1110 1010	complemento a 2

Le due differenze sono calcolate in Tabella B.3 come somma del minuendo con il complemento a 2 del sottraendo.

sottrazione normale	somma del complemento	sottrazione normale	somma del complemento
+30	0001 1110	+19	0001 0011
−22	1110 1010	−22	1110 1010
+8	0000 1000	−3	1111 1101

Tabella B.3 Sottrazione di 22 da 30 e da 19. La somma della seconda colonna dà riporto (non indicato). Il riporto segnala che il risultato è positivo, come del resto si evince dallo 0 nella posizione più significativa del risultato. La somma della quarta colonna non dà riporto e il numero risultante è negativo, come del resto si evince dall'1 nella posizione più significativa del risultato.

Da ultimo notiamo che se si sommano due numeri della stessa grandezza, uno positivo e uno negativo in complemento a 2, si ottiene lo zero (positivo e unico).

0 111	+	(+7)
1 001	=	(−7)
0000		(+0)

Convenzione

D'ora in avanti quando si parla di numeri binari negativi si intende sempre che essi sono rappresentati in complemento a 2.
Inoltre, per i numeri binari e per quelli decimali, a meno di situazioni ambigue, si omette di rappresentare la base

B.4.2 Moltiplicazione con numeri negativi

Abbiamo visto che la moltiplicazione di numeri binari senza segno si esegue col medesimo metodo dell'aritmetica decimale (Figura B.3). La moltiplicazione tra un moltiplicando negativo e un moltiplicatore positivo si esegue pure allo stesso modo estendendo il segno dei termini intermedi quanto serve a occupare la dimensione finale del risultato[3], ovvero a propagare il segno fino al bit più significativo del risultato. In Figura B.5 viene mostrato un esempio con riferimento al prodotto -13×22 rappresentati per brevità su 6 bit. Si verifichi che il risultato è effettivamente -286.

[3] A tale proposito si osservi che se si moltiplicano due numeri di n bit il risultato sarà un numero di $2n$ bit. Per esempio, una macchina a 32 bit nella quale si effettui il prodotto tra il contenuto di due registri produrrà un numero su 64 bit.

```
       1 1 0 0 1 1  × 0 1 0 1 1 0        (-13× 22)
0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 0 0 1 1
 1 1 1 1 1 0 0 1 1
 0 0 0 0 0 0 0 0
 1 1 1 0 0 1 1
 1 1 0 1 1 1 0 0 0 1 0                  (-286)
```

Figura B.5 Moltiplicazione tra moltiplicando negativo e moltiplicatore positivo. L'esempio mostra due numeri su 6 bit: 110011 (-13) e 010110 (22). Il segno dei termini intermedi è stato esteso fino ad occupare la dimensione massima contenibile in 12 bit.

Il metodo non funziona se il moltiplicatore è negativo (si vedano gli esercizi B.17 e B.18). Pertanto, se il moltiplicatore è negativo e il moltiplicando positivo, occorre complementare i due ed effettuare l'operazione come sopra. In caso di coppia di numeri negativi, si possono complementare entrambi, eseguire la moltiplicazione e tornare alla forma positiva. Un modo per evitare le complementazioni consiste nell'impiegare l'algoritmo di Booth, che tratta uniformemente i numeri in complemento a 2, positivi o negativi.

B.4.3 Algoritmo di Booth

Come visto con l'esempio di Figura B.3, in binario la moltiplicazione di due numeri si traduce in una sequenza di operazioni di somma e scorrimento (del moltiplicando). Nei primi calcolatori lo scorrimento era un'operazione molto più rapida della somma, per cui ottimizzare la moltiplicazione significava minimizzare il numero delle somme necessarie a compierla: è questo ciò che fa l'algoritmo inventato da A. D. Booth nel 1951.

Booth osservò come un numero N costituito da una sequenza di n bit ($b_{n-1}...b_0$) tutti a 1 potesse essere espresso come la differenza tra due numeri di *n+1* bit, di cui il primo col bit $b_n = 1$ seguito da tutti zeri e il secondo di tutti zeri tranne b_0, cioè come dire $N = (N + 1) - 1$. Ad esempio, 111 viene espresso come:

$$111 = 1000 \text{ - } 0001$$

Questa trasformazione permette di rimpiazzare un moltiplicatore costituito da k uni, in due termini (che chiameremo *sommatore* e *diminutore*) contenenti ciascuno un solo 1. Dunque, in luogo di k somme, sono sufficienti una somma e una sottrazione rimanendo invariato il numero di scostamenti necessari.

È facile verificare che nel caso di un moltiplicatore costituito da più raggruppamenti di uni (intervallati da uno o più zeri) il procedimento può essere applicato a ciascun raggruppamento considerato separatamente. Ad esempio

$$1\,110\,0111 = 1\,0000\,0000 \text{ - } 0\,0010\,0000 + 0\,0000\,1000 \text{ - } 0\,0000\,0001$$

quindi, trattasi sempre di una somma e di una sottrazione per ciascun raggruppamento di uni.

Per eseguire la trasformazione del moltiplicatore $b_{n-1}...b_0$ conviene procedere nel modo seguente:

a) giustapporre un bit (b_{-1}) alla destra di b_0 del moltiplicatore e porlo a 0;

b) col moltiplicatore appena modificato procedere da (b_{-1}) verso il b_{n-1} e: (i) porre 1 nel corrispondente bit del diminutore se si incontra la transizione da 0 a 1; (ii) porre 1 nel corrispondente bit del sommatore se si incontra la transizione da 1 a 0; (iii) porre 0, sia nel diminutore che del sommatore, per tutte le transizioni da 0 a 0 o da 1 a 1.

Esempio

Sia dato il numero 00110110 si determini qual è il corrispondente moltiplicatore di Booth. Successivamente si effettui il prodotto dei due numeri binari interi positivi 00110 e 01110 (6_{10} e 14_{10}) rappresentati su 5 bit.
Per quanto si riferisce al moltiplicatore 00110110 si ha:

		b_6	b_5	b_4	b_3	b_2	b_1	b_0	b_{-1}
Moltiplicatore		0	0	1	1	0	1	1	0
Diminutore	−	0	0	0	1	0	0	1	
Sommatore	+	0	1	0	0	1	0	0	
Moltiplicatore di Booth		0	+1	0	−1	+1	0	−1	

Per quanto si riferisce al prodotto 00110× 01110 ($6_{10} \times 14_{10} = 84$), il moltiplicatore di Booth è presto ottenuto.
Moltiplicatore: 0 1 1 1 0 $\Rightarrow$ moltiplicatore di Booth: +1 0 0 −1 0

Il prodotto con il moltiplicatore di Booth si esegue come qui di seguito.

Moltiplicando		0 0 1 1 0 ×	(6)
Moltiplicatore di Booth		+1 0 0 −1 0	
		0 0 0 0 0	
	−	0 0 1 1 0	
		0 0 0 0 0	
		0 0 0 0 0	
	+	0 0 0 1 1 0	
		0 0 0 1 0 1 0 1 0 0	(84)

Si può provare che l'algoritmo di Booth tratta uniformemente numeri positivi o negativi in complemento a 2, producendo un prodotto di $2n$ bit a partire da due numeri in complemento a 2 di n bit. Nel caso di numeri negativi, la relativa sequenza di Booth ha a sinistra solo zeri. Ad esempio "$1110x..x$", si traduce in "$00-1zz..z$".

Tuttavia non è necessario passare attraverso il moltiplicatore di Booth, in quanto, per come esso è ottenuto, risulta possibile operare direttamente sommando e sottraendo il moltiplicando opportunamente scostato. In pratica, l'algoritmo di Booth, per tutti i bit b_i del moltiplicatore, confronta b_i con b_{i-1} $(b_{-1} = 0)$ e svolge queste azioni

- se $(b_i - b_{i-1}) = 0$ produce un termine di zeri;
- se $(b_i - b_{i-1}) = +1$ sottrae il moltiplicando;
- se $(b_i - b_{i-1}) = -1$ somma il moltiplicando.

Si può verificare che applicando questo metodo al precedente prodotto si ottiene esattamente lo stesso risultato

Esempio

Con il metodo appena descritto, il prodotto 00110× 01110 si effettua come segue.

```
              0 0 1 1 0  ×  0 1 1 1 0   (6× 14)
              0 0 0 0 0
−           0 0 1 1 0
          0 0 0 0 0
        0 0 0 0 0
+     0 0 0 1 1 0
      0 0 0 1 0 1 0 1 0 0               (84)
```

In ambedue gli esempi precedenti la sottrazione è stata effettuata come da tabellina di Figura B.2. Visto che la sottrazione si esegue come somma del complemento a 2, si può sostituire al termine da sottrarre il suo complemento a 2 (estendendone opportunamente il segno), riducendo il tutto a operazioni di sola somma.

In Figura B.6 la tecnica appena descritta viene applicata al caso dei prodotti 6×-14 e -6×-14; si lascia al lettore il caso -6×14.

```
          0 0 1 1 0 × 1 0 0 1 0   (6×-14)
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 1 0
0 0 0 0 0 1 1 0
0 0 0 0 0 0 0
1 1 1 0 1 0
1 1 1 0 1 0 1 1 0 0               (-84)
```

```
          1 1 0 1 0 × 1 0 0 1 0   (-6×-14)
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0
1 1 1 1 1 0 1 0
0 0 0 0 0 0 0
0 0 0 1 1 0
0 0 0 1 0 1 0 1 0 0               (84)
```

Figura B.6 Esempi di moltiplicazioni col metodo di Booth, con moltiplicatore negativo. Nel prodotto di sinistra il secondo e il quinto prodotto parziale sono il complemento a 2 di 00110 con estensione del bit di segno; nel prodotto di destra i prodotti parziali nelle medesime posizioni sono il complemento a 2 di 11010, sempre con estensione del bit di segno.

Ovviamente l'algoritmo di Booth dà buoni risultati in presenza di forti raggruppamenti di 1 o di 0. Nel caso di zeri e uni alternati richiede più operazioni del procedimento di Figura B.3.

B.5 Numeri frazionari

Fino a questo punto abbiamo considerato solo numeri interi. La stringa:

$$,b_{-1}b_{-2}\cdots b_{-m}$$

si interpreta come: $b_{-1}B^{-1} + ... + b_{-m}B^{-m}$. La conversione di base si attua nel solito modo.

Conversione da binario a decimale
Sia dato il numero binario $0,101$. La conversione a decimale è banale.

$$1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = \frac{1}{2} + \frac{1}{8} = 0,5 + 0,125 = 0,625$$

Conversione da decimale a binario
La conversione da decimale a binario si ottiene con questo ragionamento: dato il numero frazionario F in base 10, si tratta di trovare la stringa $b_{-1}b_{-2}\cdots b_{-m}$ tale per cui:

$$F = b_{-1}b_{-2}\cdots b_{-m}$$

Osservando che b_{-1} è la parte intera del prodotto

$$2 \times F = b_{-1} + b_{-2} \times 2^{-1} + \cdots b_{-m} \times 2^{-(m-1)}$$

si deduce che la ricerca dei coefficienti b_i richiede un processo di successive moltiplicazioni della parte frazionaria con estrazione della parte intera. Il processo termina quando la parte frazionaria risulta 0 (oppure non termina se il numero è periodico).

Esempio

Si convertano in forma binaria i numer1 decimali $0,78125$ e 0,9.
Per il numero $0,78125$ l'applicazione del metodo porta ad effettuare queste operazioni:

$$\begin{aligned} 0,78125 \times 2 = 1,5625 &\quad \rightarrow 1 \\ 0,5625 \times 2 = 1,125 &\quad \rightarrow 1 \\ 0,125 \times 2 = 0,250 &\quad \rightarrow 0 \\ 0,25 \times 2 = 0,5 &\quad \rightarrow 0 \\ 0,5 \times 2 = 1,0 &\quad \rightarrow 1 \end{aligned}$$

dunque $(0,78125)_{10} = (0,11001)_2$.

Per il numero 0,9, si ha:

$$\begin{aligned} 0,9 \times 2 = 1,8 &\quad \rightarrow 1 \\ 0,8 \times 2 = 1,6 &\quad \rightarrow 1 \\ 0,6 \times 2 = 1,2 &\quad \rightarrow 1 \\ 0,2 \times 2 = 0,4 &\quad \rightarrow 0 \\ 0,4 \times 2 = 0,8 &\quad \rightarrow 0 \\ 0,8 \times 2 = 1,6 &\quad \rightarrow 1 \\ 0,6 \times 2 = 1,2 &\quad \rightarrow 1 \\ 0,2 \times 2 = 0,4 &\quad \rightarrow 0 \\ 0,4 \times 2 = 0,8 &\quad \rightarrow 0 \\ \cdots\cdots\cdots\cdots & \end{aligned}$$

dunque $(0,9)_{10} = (0,1\,1100\,1100)_2$ periodico.

B.5.1 Numeri in virgola fissa

Per i numeri interi si assume che la virgola sia posizionata all'estrema destra. In modo del tutto analogo per i numeri frazionari la virgola viene considerata all'estrema sinistra, come nei due numeri dell'esempio precedente. Un generico numero N sarà formato da una parte intera e da una parte frazionaria separate tra loro dalla virgola.

Quando la virgola separa la parte intera dalla parte frazionaria si parla di notazione in *virgola fissa*.

Esempio

Convertire il numero decimale 23,59375 in forma binaria.
Per il 23_{10}, la serie successiva di quozienti e resti ottenuti dividendo per 2 è:

$$(11,1),(5,1),(2,1),(1,0)(0,1)$$

Prendendo i resti da destra verso sinistra $23_{10} = 10111_2$

Mentre per $,59375_{10}$

$$\begin{aligned} 0,59375 \times 2 = 1,1875 &\quad \to 1 \\ 0,1875 \times 2 = 0,375 &\quad \to 0 \\ 0,375 \times 2 = 0,75 &\quad \to 0 \\ 0,75 \times 2 = 1,5 &\quad \to 1 \\ 0,5 \times 2 = 1,0 &\quad \to 1 \end{aligned}$$

dunque $0,59375_{10} = 0,10011_2$
Ne consegue

$$23,59375_{10} = 10111,10011_2$$

B.6 Numeri in virgola mobile

Frequentemente, e in modo particolare nei problemi di calcolo tecnico e scientifico, si ricorre a rappresentazioni normalizzate aventi lo scopo di sollevare l'utilizzatore dai problemi connessi con il controllo della posizione della virgola e con l'aumento del numero di cifre a seguito delle operazioni aritmetiche che via via vengono eseguite.

Nel calcolo numerico i dati vengono di solito espressi come prodotto di due fattori, il primo dei quali comprende le cifre significative del numero da rappresentare mentre il secondo è una potenza del 10, il cui esponente definisce la posizione della virgola nel numero.

In generale si può dire che un dato numerico qualsiasi ammette una rappresentazione approssimata come la seguente:

$$\pm x_1 x_2 x_3 ... x_h, y_1 y_2 y_3 ... y_k \times B^{\pm a_1 a_2 ... a_n}$$

dove B è la base del sistema di numerazione, $x_1 x_2 x_3 ... x_h$, $y_1 y_2 y_3 ... y_k$ e $a_1 a_2 ... a_n$, sono cifre dello stesso sistema.

Il numero $x_1 x_2 x_3 ... x_h, y_1 y_2 y_3 ... y_k$ viene chiamato mantissa, mentre il numero $a_1 a_2 ... a_n$ viene chiamato esponente o caratteristica.

Esempio

Rappresentare in notazione scientifica i numeri 127 000 000 e 0,0000045.

Il numero 127 000 000 può essere scritto nella forma 127×10^6; il numero 0,0000045 può essere scritto come 45×10^{-7}. Ovvero

$$\begin{aligned} x_1 x_2 x_3, y_1 y_2 y_3 = 127,000 &\qquad a_1 = 6 &\qquad B = 10 \\ x_1 x_2 x_3, y_1 y_2 y_3 = 45 &\qquad a_1 = -7 &\qquad B = 10 \end{aligned}$$

B.6.1 Rappresentazione normalizzata

La normalizzazione richiede che sia definita la posizione della virgola. Possiamo, ad esempio, imporre che la prima cifra significativa si trovi immediatamente a destra della virgola. A tal fine basta aumentare o diminuire il valore dell'esponente di tante unità quante sono le posizioni di cui è stato spostata la virgola[4]. La forma ottenuta con questa convenzione è detta *rappresentazione esponenziale normalizzata*.

Esempio

Rappresentare i numeri dell'esempio precedente in forma scientifica normalizzata.

I due numeri vengono così trasformati:

$$\begin{aligned} 127 \times 10^6 &= 0,127 \times 10^9 \\ 45 \times 10^{-7} &= 0,45 \times 10^{-5} \end{aligned}$$

La precedente rappresentazione esponenziale normalizzata contiene ancora caratteri ridondanti ed è possibile, mediante ulteriori convenzioni, arrivare a una rappresentazione più compatta. Facendo riferimento alla notazione decimale, si può stabilire di

- eliminare i caratteri non necessari, ovvero:
  - lo zero che indica la parte intera della mantissa;
  - la virgola decimale;
  - il segno di prodotto;
  - il valore della base della potenza;
- prefissare il un numero di cifre della mantissa;
- limitare l'esponente ai valori compresi in un opportuno intervallo;
- utilizzare un esponente convenzionale (polarizzato) ottenuto sommando all'esponente effettivo una costante di polarizzazione (*bias*) che lo renda sempre positivo, eliminando quindi il segno dell'esponente[5];
- disporre i tre elementi rimasti (segno, esponente polarizzato e mantissa) in un ordine stabilito. L'ordine standard è quello sottostante.

s (segno)	esp (esponente)	M (mantissa)

Esempio

Stabiliamo questa convenzione

- lunghezza mantissa: 8 cifre;
- valore effettivo dell'esponente: da -50 a $+49$;
- valore della costante di polarizzazione: 50;

i numeri dell'Esempio precedente, pag. 15, si rappresentano nel modo seguente:

[4]Si noti che questa convenzione, riportata alla rappresentazione binaria, è diversa da quella dello standard IEEE descritto al Paragrafo B.7, che invece prevede che a sinistra della virgola ci sia un 1 anziché uno 0.

[5]Indicando con C la costante di polarizzazione, si usa la dizione numero in "eccesso C".

		s	esp	M
$0{,}127\times10^{9}$	:	+	59	12700000
$0{,}15\times10^{-5}$	:	+	45	15000000

Torniamo al sistema binario. In un calcolatore, il segno richiede un bit, mentre per esponente e mantissa si tratta di scegliere misure convenienti, con il vincolo che i tre componenti stiano in una misura predefinita, normalmente in una parola (32 bit) o in una doppia parola. Su 32 bit le dimensioni standard dei campi sono quelle di Figura B.7.

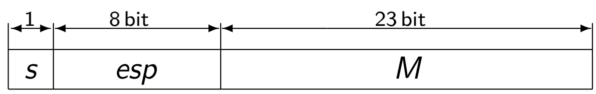


Figura B.7 Formato di un numero in virgola mobile su 32 bit. Questo formato corrisponde al formato in singola precisione dello standard IEEE.

- Il primo bit rappresenta il segno della mantissa (0 per il segno +, 1 per il segno−).
- Gli 8 bit successivi rappresentano l'esponente polarizzato. Con 8 bit a disposizione l'esponente polarizzato può variare tra 0 e 255. Quello effettivo è compreso tra −128 e +127. Se si assume la costante di polarizzazione pari a 128, il numero -128 compare come 0.
- I 23 bit di destra rappresentano il valore assoluto della mantissa in forma normalizzata.

Esempio

Si dia una rappresentazione normalizzata, secondo il formato appena descritto dell'equivalente binario del numero 204, 17437, assumendo che la convenzione preveda lo 0 (nascosto) a sinistra della virgola e 1 subito a destra.

Al numero 204, 17437 corrisponde il binario 11001100, 00101100101111.
Per effettuare la normalizzazione si deve far scorrere la virgola di 8 posizioni a sinistra, in modo da portare il primo 1 subito dietro la virgola, e corrispondentemente moltiplicare per 2. Ne consegue che l'esponente effettivo (della base 2) è 8, ovvero 1000 in forma binaria.

Si ha dunque:

- bit di segno: 0;
- esponente effettivo: 0000 1000;
- mantissa: 110 0110 0001 0110 0101 1110

All'esponente effettivo si somma la costante di polarizzazione $(128)_{10}$, pari a $(1000\,0000)_2$ e si ottiene questa rappresentazione:

		s	esp	M
204, 17437	:	0	1000 1000	110 0110 0001 0110 0101 1110

La rappresentazione in virgola mobile può dar luogo al fenomeno del traboccamento. Quando il valore dell'esponente supera il massimo previsto dalla rappresentazione si ha un *overflow*, mentre quando tale valore diventa più piccolo del minimo previsto si ha un *underflow*. L'esponente non può essere troncato altrimenti variano gli ordini di grandezza. Al contrario la mantissa può essere troncata o approssimata; ciò influisce solo sulla precisione (Paragrafo B.7.2).

Fino agli anni ottanta (XX secolo) ogni costruttore tendeva a dare una propria rappresentazione ai numeri in virgola mobile. Nel 1985 è stato definito lo standard IEEE-754 (Paragrafo B.7) al quale si è conformata la produzione. Ciò non toglie che non ci siano costruttori che, pur rispettando lo standard, quando sussistono ragioni di compatibilità, continuino a utilizzare anche i propri formati proprietari.

B.6.2 Operazioni in virgola mobile

La descrizione dettagliata degli algoritmi con cui vengono effettuate le operazioni algebriche in virgola mobile non rientra nei fini di questa trattazione. Per ulteriori approfondimenti si rimanda alla letteratura [Omo94], [HP06]qui di seguito, per dare un'idea di come vengono svolte tali operazioni, sono riportati in forma molto semplificata i principali passi dell'algoritmo di somma/sottrazione, moltiplicazione e divisione.

Somma/sottrazione Le operazioni di somma e sottrazione di numeri in virgola mobile richiedono preliminarmente che gli esponenti dei due addendi siano uguali. A tal fine occorre traslare le mantisse dei due numeri, una rispetto all'altra, in modo da riportarli allo stesso esponente. La regola per l'addizione e la sottrazione in virgola mobile può essere riassunta nei seguenti passi.

1. Si trasla a destra la mantissa del numero con l'esponente minore per un numero di bit pari alla differenza degli esponenti, in modo da rendere questi ultimi uguali.
2. Si pone l'esponente del risultato uguale all'esponente (del più grande).
3. Si effettua l'addizione o la sottrazione delle mantisse e si determina il segno del risultato.
4. Si normalizza il risultato se necessario.

Moltiplicazione Per la moltiplicazione non è necessario l'allineamento delle mantisse. L'algoritmo può essere riassunto come qui di seguito. Il dettaglio è mostrato in Figura B.8.

1. Si sommano gli esponenti e si sottrae la costante di polarizzazione (la somma degli esponenti raddoppia la costante di polarizzazione).
2. Si dividono le mantisse e si determina il segno del risultato.
3. Si normalizza il risultato se necessario.
4. Il segno si ottiene come somma (modulo 2) dei segni dei due termini.

Divisione La divisione è simile alla moltiplicazione.

1. Si sottraggono gli esponenti e si somma la costante di polarizzazione.
2. Si moltiplicano le mantisse e si determina il segno del risultato.
3. Si normalizza il risultato se necessario.
4. Il segno è dato dalla somma (modulo 2) dei segni dei due termini.

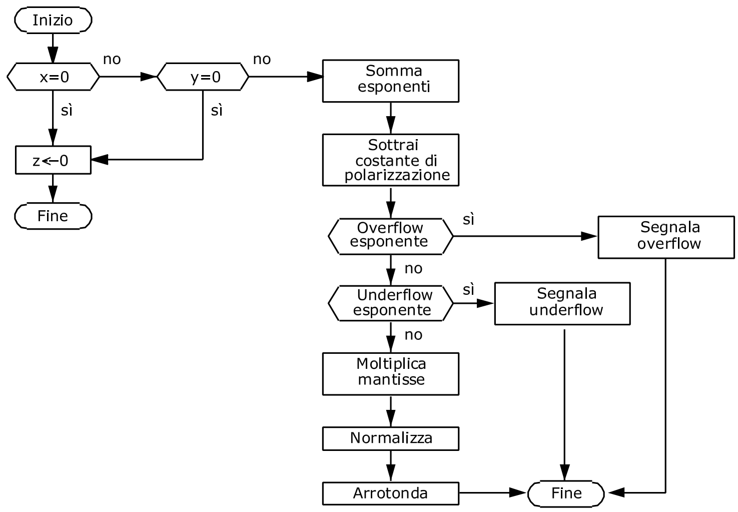


Figura B.8 Moltiplicazione di due numeri in virgola mobile ($z \leftarrow x \times y$).

Esempio

Dati i due numeri positivi 111,00101 e 10,1, rappresentarli in forma normalizzata, assumendo che l'esponente sia su 4 bit e la mantissa su 12 bit, la costante di polarizzazione sia 16 e che per la mantissa valga la convenzione fatta in precedenza, cioè 0 nascosto. Successivamente si effettui i prodotto e si rinormalizzi. Si verifichi risultato con la numerazione decimale.

Cominciamo osservando che la normalizzazione comporta lo scorrimento verso destra di 3 posizioni per il primo numero e di 2 per il secondo. Dunque gli esponenti effettivi risultano 011 e 10 rispettivamente. Essendo 1000 la costante di polarizzazione i due esponenti polarizzati sono 1011 e 1010. La rappresentazione normalizzata e questa

	s	esp	M
111,00101 :	0	1011	1110 0101 0000
10,10000 :	0	1010	1010 0000 0000

Se si effettua il prodotto, l'esponente polarizzato diventa 1101 (esponente effettivo 101), mentre il prodotto delle mantisse dà 0,100011110010 e quindi la normalizzazione non richiede ulteriore aggiornamento dell'esponente (che resta pari a 5). Per quanto riguarda il segno, la somma dei due segni $(0 + 0)$ dà zero, come deve essere poiché il risultato è positivo. Si ha dunque

s	esp	M
0	1101	1000 1111 0010

Facciamo ora la verifica. Anzitutto il numero binario 111,00101 corrisponde al numero decimale $1\times 2^2+1\times 2^1+1\times 2^0+1\times 2^{-3}+1\times 2^{-5} = 4+2+1+0,125+0,03125 = 7,15625$, mentre al numero binario 10,1 corrisponde il numero decimale $1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} = 2 + 0,5 = 2,5$.

Il loro prodotto $(7,15625 \times 2,5)$ fa $17,890625$.

Vediamo ora cosa dà la rappresentazione normalizzata. Cominciamo col convertire la mantissa

$$\begin{aligned} M &= 1 \times 2^{-1} + 1 \times 2^{-5} + 1 \times 2^{-6} + 1 \times 2^{-7} + 1 \times 2^{-8} + 1 \times 2^{-11} = \\ &= 0,5 + 0,03125 + 0,015625 + 0,007812500000 + 0,003906250000 + \\ &\quad +0,000488281250 = 0,559082031250 \end{aligned}$$

Se ora moltiplichiamo questo valore per 2^5 otteniamo il risultato che già conosciamo

$$0,559082031250 \times 32 = 17,890625$$

L'esempio appena concluso ci consente di fare un'osservazione sulla precisione. Avevamo assunto che la mantissa fosse su 12 bit. Il prodotto delle mantisse è risultato su 11 bit, quindi non c'è stato nessun problema di troncamento/precisione. Se le mantisse fossero state date su 10 bit, il risultato del prodotto (0,100011110010) non sarebbe stato perfettamente rappresentabile. Troncando le ultime due cifre, avremmo avuto una mantissa risultante pari a 10 0011 1100, cui corrisponde $17,875$ invece di $17,890625$, con uno scarto non indifferente già sulla seconda cifra dopo la virgola. In conclusione, la rappresentazione in virgola mobile consente di estendere il campo dei numeri rappresentabili, ma a scapito della precisione. A tale proposito si veda quanto detto ai successivi paragrafi B.7.1, pag. 21, e B.7.2.

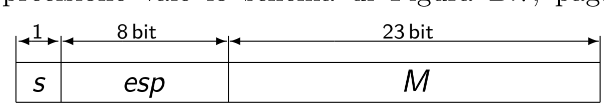
B.7 Standard IEEE 754-1985 per l'aritmetica binaria in virgola mobile

Per eliminare la confusione dovuta alle differenze tra i diversi formati proprietari, relativamente al numero di bit usati per rappresentare l'esponente e la mantissa, all'intervallo di esistenza degli esponenti, ai metodi di arrotondamento e al trattamento delle eccezioni (per esempio, l'overflow), è stato introdotto lo standard IEEE 754-1985 [IEE85].

Lo standard adotta criteri di rappresentazione simili a quelli visti nel Paragrafo B.6.1. Sostanzialmente esso definisce

- un formato in singola precisione
- un formato in doppia precisione
- un formato esteso

Per la singola precisione vale lo schema di Figura B.7, pag. 16 che qui riportiamo.



Il segno occupa sempre il bit più significativo. Per la doppia precisione ci sono 11 bit per l'esponente e 52 per la mantissa; nelle macchine a 32 bit la doppia precisione richiede due parole di memoria.

Il formato è definito da tre parametri:

- P : precisione, ovvero numero di bit che compongono la mantissa;
- E_{max}: esponente massimo effettivo;

- E_{min}: esponente minimo effettivo.

Tralasciando per il momento il formato esteso, i valori dei parametri sono quelli di Tabella B.4.

	Singola	Doppia
P (bit)	23	52
E_{max}	127	1023
E_{min}	−126	−1022
Costante di polarizzazione	127	1023
Ampiezza della parola (bit)	32	64
Ampiezza esponente (bit)	8	11

Tabella B.4 Parametri dello standard IEEE 754-1985 per la singola e doppia precisione.

Lo standard impiega esponenti polarizzati. La costante di polarizzazione è 127 nel caso di singola precisione e 1023 nel caso di doppia precisione.

Il campo della mantissa si compone della sola parte frazionaria, ma, diversamente da quanto ipotizzato al Paragrafo B.6.1, pag. 15, dove si era assunto che la parte intera della mantissa fosse 0, lo standard IEEE stabiisce che la parte intera sia 1. Questo 1 è implicito e non si rappresenta (si dice che il bit è nascosto).

Esempio

Convertire il numero 204, 17437 dell'esempio di pagina 16 nel formato IEEE.

Al numero 204, 17437 corrisponde il binario 11001100, 00101100101111.
Questa volta la virgola va fatta scorrere di 7 posizioni dovendo portare a 1 la parte intera. Ne consegue che l'esponente effettivo è 111 (in binario). L'esponente eccesso 127 diventa $7 + 127 = 134$ ovvero $111 + 0111\,1111 = 1000\,0110$. La mantissa diventa 100 1100 0010 1100 1011 1100. In conclusione, la rappresentazione binaria del numero 204, 17437 nello standard IEEE è quella che segue.

0 1000 0110 100 1100 0010 1100 1011 1100

Si osservi che questa rappresentazione è diversa da quella cui si è pervenuti nell'esercizio di pagina 16, a causa della differente assunzione sulla parte intera della mantissa.

La Tabella B.4 definisce i valori E_{min} ed E_{max} entro cui può variare l'esponente effettivo E. Da questi valori consegue che l'esponente polarizzato esp varia tra 1 e 254 nel caso di singola precisione e tra 1 e 2046 in caso di doppia precisione.

Conviene dire due parole sul motivo per cui la costante di polarizzazione è stata scelta pari a 127 (numero massimo rappresentabile in 7 bit), anziché 128. Il valore $esp = 0$ (ovvero $E = E_{min} - 1 = -127$) viene usato per codificare ±0 e i numeri denormalizzati. I valori $esp = 255$, per la singola precisione, ed $esp = 2047$, per la doppia (ovvero $E = E_{max} + 1$), sono usati per codificare $\pm\infty$ e i cosiddetti NaN (*not-a-number*)[6].

[6] Un NaN è un'entità simbolica codificata in virgola mobile, che serve, per esempio, a dare valore a una variabile non inizializzata, in modo che il suo uso determini un'eccezione. Si osservi che lo zero e l'infinito hanno due rappresentazioni.

Interpretazione dei campi
Il valore (v) di un numero in virgola mobile è dedotto dagli elementi componenti i campi secondo l'interpretazione di Tabella B.5. Notare che l'espressione $(-1)^s$ dà 1 (ovvero segno positivo) se $s = 0$, dà -1 (ovvero segno negativo) se $s = 1$.

	esp	M	v
Numero normalizzato	$0 < esp < 255$	qualunque	$v = (-1)^s(1, M)2^{esp-127}$
Numero denormalizzato	$esp = 0$	$M \neq 0$	$v = (-1)^s(0, M)2^{-126}$
Zero	$esp = 0$	$M = 0$	$v = (-1)^s 0$
Infinito	$esp = 255$	$M = 0$	$v = (-1)^s \infty$
NaN	$esp = 255$	$M \neq 0$	$v =$ NaN
Numero normalizzato	$0 < esp < 2047$	qualunque	$v = (-1)^s(1, M)2^{esp-1023}$
Numero denormalizzato	$esp = 0$	$M \neq 0$	$v = (-1)^s(0, M)2^{1022}$
Zero	$esp = 0$	$M = 0$	$v = (-1)^s 0$
Infinito	$esp = 2047$	$M = 0$	$v = (-1)^s \infty$
NaN	$esp = 2047$	$M \neq 0$	$v =$ NaN

Tabella B.5 Interpretazione dei numeri in virgola mobile nello standard IEEE 754. In alto singola precisione, in basso doppia precisione.

Spieghiamo ora il motivo per il quale sono stati previsti i *numeri denormalizzati*. Per convenzione (Tabella B.5) questi numeri hanno $esp = 0$ e 0 come bit implicito a sinistra della virgola[7]. Il più piccolo numero normalizzato in singola precisione si ha con $M = 0$ ed $esp = 1$, ovvero $1,0 \times 2^{-126}$. I numeri denormalizzati servono a esprimere quantità inferiori a questa. Infatti, il minimo numero denormalizzato si ha quando la mantissa contiene tutti 0 eccetto un 1 nell'ultimo bit a destra, cioè 2^{-23} (singola precisione). Poiché l'esponente effettivo è -126, ne consegue che il numero minimo rappresentabile con la forma denormalizzata è pari a 2^{-149}, molto più piccolo del precedente. Il vantaggio offerto dai numeri denormalizzati si manifesta quando il calcolo porta a numeri molto piccoli. Senza i numeri denormalizzati si passerebbe da $1,0 \times 2^{-126}$ a zero; essi consentono di riempire l'intervallo $(2^{-126} - 0)$ con 2^{23} numeri diversi[8].

Si noti che lo standard prevede anche la rappesentazione dell'infinito e di entità che non sono numeri. L'uno e l'altro possono essere usati come operandi. Ovviamente se si somma un numero a infinito il risultato è infinito. Se si somma un numero con un NAN il risultato è una segnalazione (eccezione) di non validità; se si divide infinito per infinito o si moltiplica infinito per zero il risultato è un NAN.

B.7.1 Formato esteso

Lo standard prevede anche due formati estesi (singola e doppia precisione). Questi sono definiti in maniera piuttosto lasca, nel senso che, per esempio, per le dimensioni dei campi vengono dati solo i limiti inferiori. In pratica, i due effettivi standard sono quelli menzionati in precedenza e ogni costruttore è sostanzialmente libero di farsi una propria

[7] Essi si distinguono dai normalizzati proprio dall'esponente; si distinguono dallo 0 per avere $M \neq 0$.

[8] Si noti che il massimo numero denormalizzato ha $esp = 0$ e tutti 1 nella mantissa (vedere Tabella B.5, pari circa a $0,9999999 \times 2^{-126}$ (sostanzialmente uguale al più piccolo numero normalizzato rappresentabile).

rappresentazione estesa. In pratica lo standard recepiva lo stato di fatto. Per esempio, l'IBM con, il sistema S/370, aveva introdotto da molto tempo un suo formato esteso a 128 bit.

È interessante il caso dell'architettura ×86. Nel 1980, cinque anni prima della stesura finale dello standard IEEE 745, l'Intel introdusse il coprocessore matematico 8087, un dispositivo che, operando in modo sincronizzato con la CPU 8086, estendeva il repertorio delle istruzioni di quest'ultima con le operazioni in virgola mobile, aggiungendo circa 60 nuove istruzioni. L'8087 impiegava le due rappresentazioni – in singola e doppia precisione – che poi sarebbero state sostanzialmente adottate nello standard IEEE, ma definiva anche una rappresentazione estesa su 80 bit. Questo formato, che prevede una mantissa di 68 bit – l' esponente rimane a 11 – viene usata internamente al dispositivo per aumentare la precisione dei calcoli. A tale scopo, I numeri vengono convertiti su 80 bit quando vengono caricati nei registri interni del dispositivo e convertiti nel formato standard quando il contenuto dei registri viene copiato in memoria. Sono anche previste istruzioni per scambiare numeri in formato esteso da/verso la memoria, utili quando lo svolgimento del calcolo richieda eventuali salvataggi temporanei dei contenuti dei registri. In ogni caso le operazioni in virgola mobile vengono effettuate con mantissa di 68 bit, conferendo maggior precisione ai calcoli. A partire dal 486 il coprocessore matematico è stato integrato in tutti i modelli successivi di CPU dell'architettura ×86.

B.7.2 Precisione

Bit di guardia

Nell'esecuzione delle operazioni in virgola mobile (Paragrafo B.6.2) gli esponenti e le mantisse degli operandi vengono caricati nei registri di CPU. Normalmente i registri sono più ampi della mantissa. Per esempio: nel caso del formato IEEE singola precisione la mantissa è su 23 bit più il bit nascosto.

I rimanenti 8 bit, di un eventuale registro a 32 bit, possono essere usati come *bit di guardia*. Questi bit vengono posti a zero quando la mantissa è caricata nel registro e vengono impiegati nei passaggi intermedi dell'elaborazione, ai fini di una maggior precisione, in quanto consentono una rappresentazione più estesa. Quando il numero viene normalizzato, prima di essere ricopiato in memoria, si richiede l'arrotondato secondo le regole sotto esposte.

Arrotondamento

Quando si esegue un'operazione su due numeri in virgola mobile, di solito, il risultato è un valore che non si può rappresentare in modo esatto.

Con riferimento alla numerazione decimale, si considerino i numeri $2,1$ e $0,5$. Essi sono rappresentabili su due cifre, ma il loro prodotto ($2,1 \times 0,5 = 1,05$) non lo è. Se si vuole rappresentare il risultato del prodotto su due cifre si pone questa domanda: si arrotonda a $1,1$ oppure a $1,0$? Si noti che questa sarebbe una situazione ambigua: di per sé non c'è motivo per l'una o l'altra scelta. In una situazione ambigua lo standard IEEE prevede l'arrotondamento al valore la cui cifra meno significativa è pari (vedere qui di seguito). Quindi, il nostro numero decimale 1.05, seguendo le regole dello standard, verrebbe arrotondato a 1.0.

In generale lo standard IEEE 754-1985 prevede quattro modalità di arrotondamento:

- arrotondamento verso lo zero (il risultato viene troncato);
- arrotondamento al valore più vicino (modalità standard);

- arrotondamento verso $+\infty$ (il risultato è arrotondato verso verso l'alto) ;
- arrotondamento a $-\infty$ (il risultato è arrotondato verso il basso).

L'arrotandamento verso lo zero, corrisponde al brutale troncamento. Questo metodo è certamente il più rapido, ma ha il grave difetto di introdurre un errore che va da 0 a 1 sulla cifra meno significativa, asimmetrico rispetto allo zero. Il valore troncato è sempre minore del valore vero, introducendo una "polarizzazione" verso lo zero.

L'arrotondamento al valore più vicino può essere spiegato considerando, ad esempio, il caso del numero $0, b_1b_2b_3b_4b_5b_6b_7$ che debba essere arrotondato su tre cifre dopo la virgola, eliminando cioè gli ultimi 4 bit. Se questi 4 bit contengono un numero superiore a 1000, il modo corretto di arrotondare è aggiungere 1 all'ultimo bit rappresentabile, ovvero arrotondando al successivo numero rappresentabile. Se i 4 bit contengono un numero inferiore a 1000, il modo corretto di arrotondare è troncare, ovvero arrotondare al numero rappresentabile inferiore. In altre parole, il numero $0, b_1b_2b_3 1010$ si arrotonda a $0, b_1b_2b_3 + 0,001$, mentre il numero $0, b_1b_2b_3 0010$ si arrotonda a $0, b_1b_2b_3$.

La precedente regola non specifica il caso ambiguo (parte da arrotondare a metà) visto all'inizio del paragrafo in riferimento alla notazione decimale. Lo standard IEEE stabilisce il troncamento deve dare un risultato pari: arrotondare verso l'alto se l'ultimo bit rappresentabile è 1, oppure troncare se l'ultimo bit rappresentabile è 0.

Esempio

Arrotondare $0, b_1b_2b_3 1000$.

Notiamo anzitutto che $0, b_1b_2b_3 1000$ sta esattamente a metà tra le due possibili rappresentazioni troncate. Ne consegue che se $b_3 = 0$ il numero è troncato a 0, cioè

$$0, b_1b_2 0\,1000 \Rightarrow 0, b_1b_2 0$$

mentre se $b_3 = 1$ il numero è arrotondato al pari superiore, cioè

$$0, b_1b_2 1\,1000 \Rightarrow 0, b_1b_2 1 + 0,001$$

.

L'esempio ha reso evidente che l'arrotondamento al valore più vicino comporta un errore che approssimativamente sta nel campo $(-1/2, +1/2)$ dell'ultimo bit rappresentabile, e che può richiedere, oltre alla possibile addizione, una possibile ulteriore normalizzazione del risultato.

B.7.3 Eccezioni

Un'eccezione è il verificarsi di un evento anormale rispetto al previsto funzionamento. Il verificarsi di situazioni di eccezione viene rilevato dalla logica della CPU rendendone possibile l'esame da parte del programmatore, che può decidere quali provvedimenti adottare.

Lo standard IEEE prevede cinque cause di eccezione aritmetica:

- *underflow*
- *overflow*
- *divisione per zero*
- *eccezione di inesattezza*

- *eccezione di invalidità.*

Le eccezioni di underflow, overflow, divisione per zero, sono presenti anche in altri standard. L'eccezione di inesattezza è la caratteristica dell'aritmetica IEEE e si verifica sia quando il risultato di una operazione deve essere arrotondato, sia quando l'operazione incorre in un overflow.

Quando si verifica una di queste eccezioni, è previsto l'aggiornamento di un bit di segnalazione, ma il calcolo può proseguire. Questi segnalatori, una volta attivati, rimangono tali fino a che non vengono disattivati esplicitamente. Lo standard raccomanda (ai progettisti di calcolatori) di introdurre anche un bit di abilitazione di interruzione, uno per ciascuna eccezione. In questo modo se si verifica una delle eccezioni, e il corrispondente bit di abilitazione dell'interruzione è attivo, entra il gestore delle interruzioni predisposto dall'utente (ovviamente, in questo caso, il bit di segnalazione non è necessario).

Infine, lo standard prevede che qualora si verifichi una interruzione dovuta a un'eccezione aritmetica, si possa risalire all'operazione che l'ha generata e anche al valore dei suoi operandi.

B.8 Informazioni di carattere alfanumerico

L'informazione elaborata da un calcolatore elettronico non è solamente di carattere numerico. Infatti è necessario rappresentare anche informazioni di tipo testuale o di altro genere, codificate attraverso simboli binari. Questa necessità sorge, ad esempio, se si vuole tener traccia del nome di una persona. È naturale che all'interno del calcolatore un nome venga rappresentato in modo del tutto analogo alla forma scritta e cioè attraverso una stringa di caratteri.

Per rappresentare l'informazione in forma testuale alfanumerica si rende necessario stabilire una corrispondenza biunivoca tra caratteri e segni dell'alfabeto e configurazioni di cifre binarie. La corrispondenza in questione si dice *codifica*. Una forma molto naturale consiste nel codificare un carattere all'interno di un singolo *byte* (un raggruppamento di 8 bit).

I normali dispositivi periferici con i quali l'uomo interagisce con il calcolatore scambiano informazioni in forma codificata. Se consideriamo ad esempio un normale processo di elaborazione, questo prevederà tre fasi.

1. I parametri dell'elaborazione vengono introdotti attraverso un dispositivo di ingresso, come sequenze di caratteri, nella codifica adottata dal sistema (si veda poco oltre).
2. I parametri di tipo numerico vengono convertiti da rappresentazione alfanumerica in rappresentazione binaria per essere elaborati. Al termine dell'elaborazione i risultati vengono convertiti da rappresentazione binaria in rappresentazione alfanumerica.
3. I risultati in forma alfanumerica vengono presentati su un dispositivo di uscita.

B.8.1 Codifica ASCII

Esistono diverse forme di codifica alfanumerica. La più nota è sicuramente la codifica ASCII (*American Standard Code for the Interchange of Information*). Originariamente la codifica ASCII era su 7 bit. In un secondo tempo è stata portata a 8 bit, imponendo a zero il bit più significativo. Nella codifica originale si hanno quindi 128 possibili simboli e ciò consente di rappresentare agevolmente tutti i caratteri alfanumerici (in forma maiuscola e minuscola per gli alfabetici), i segni di punteggiatura, gli usuali simboli matematici ecc.

oltre a un buon numero di altri caratteri usati normalmente come *caratteri di controllo*. Successivamente la codifica è stata estesa a 256 codifiche, sfruttando anche il valore “1” dell'ottavo bit (ASCII esteso). A tutti i valori della codifica (eccetto lo zero) è stata associata una rappresentazione grafica. In Tabella B.6 viene riportata la codifica ASCII standard (ottavo bit a zero).

A titolo di esempio consideriamo il numero 25097. Con riferimento alla Tabella B.6, la sua rappresentazione in formato ASCII corrisponde a questa sequenza (in formato esadecimale): 32 35 30 39 37. In altre parole, il numero viene rappresentato in memoria su 5 byte consecutivi, il cui contenuto è quello appena indicato. Allo stesso modo, la rappresentazione in memoria della stringa “BLA bla” è data da questa sequenza di byte: 42 4C 41 20 62 6C 61 (si vedano gli Esercizi B.23 e B.26).

Si noti che le due colonne di sinistra di Tabella B.6 rappresentano caratteri di per sé non stampabili. Essi vengono normalmente impiegati come caratteri controllo, specialmente nei protocolli di comunicazione. Ad esempio, il carattere `SOH` (*Start of Header*) viene impiegato come carattere di inizio di un messaggio, mentre il carattere `ACK` viene impiegato all'interno dei messaggi come indicatore di *Acknowledgement*.

La codifica ASCII estesa permette di rappresentare anche i caratteri accentati, presenti in molte lingue, tra cui l'italiano, ma non previsti dalla versione standard di Tabella B.6.

Negli ultimi anni anche la codifica ASCII estesa ha cominciato ad apparire insufficiente; per questo motivo hanno fatto la loro comparsa codici su 2 byte, come la codifica *Unicode*, di cui si parla qui di seguito.

				Bit più significativi							
				0	0	0	0	0	0	0	0
				0	0	0	0	1	1	1	1
Bit meno significativi				0	0	1	1	0	0	1	1
				0	1	0	1	0	1	0	1
0	0	0	0	NUL	DLE	Spaz	0	@	P	`	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	”	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NACK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	'	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	−	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	DEL

Tabella B.6 La codifica ASCII standard. È immediato esprimere in forma esadecimale la codifica corrispondente a un dato carattere. Ad esempio, la codifica del carattere di spazio (“Spaz”) è 20, quella del carattere “A” è 41, quella del carattere “9” è 39, mentre quella del “LF” (*Line Feed*) è A.

B.8.2 Unicode e UTF-8

Appare subito evidente che la codifica ASCII, se può andar bene agli anglosassoni, che non hanno accenti, va un poco stretta per le lingue come l'italiano che hanno lettere accentate, ancor più stretta per gli alfabeti come il cirillico o, peggio ancora, l'arabo; il giapponese e cinese sono un delirio. Per ovviare alla limitazione della codifica ASCII, a suo tempo, si è formato un consorzio, denominato UNICODE, che rappresenta i caratteri su 16 bit, ogni carattere con un codice distinto. In sostanza ci sono ben 65.536 caratteri diversi. Ma neanche questi bastano, tanto che sono previsti 17 "piani" di 65.536 (al momento solo 6 piani sono assegnati).

Per ovviare allo spreco di memoria, sono state proposte le codifiche UTF (Unicode Transformation Format), in versione a 8, 16, 32 bit, che pure ricorrono ai codici UNICODE, ma ne fanno un uso più compatto. Qui di seguito discutiamo brevemente la codifica UTF-8.

Anzitutto i codici da 0 a 127 di UTF-8 corrispondono esattamente ai caratteri ASCII, di modo che essi possono essere rappresentati su 7 bit. Ne consegue che se un testo usa solo caratteri che fanno parte della codifica ASCII, basta un byte a carattere; questo byte deve avere 0 nel bit più significativo. Quando si esce fuori dal campo corrispondente all'ASCII si usano gruppi di byte per rappresentare il singolo carattere. Il primo byte di un gruppo ha un numero di 1 in posizione più significativa pari al numero totale di byte nel gruppo; i successivi byte del gruppo hanno sempre 10 a sinistra. Questa regola evita di confondere il primo byte con quelli che seguono. Teoricamente sarebbero possibili gruppi di dimensioni fino a 6 byte, ma lo standard corrente limita a 4. Del resto, con i caratteri su un byte e quelli su due si copre tutto l'alfabeto latino, il greco, il cirillico, il copto, l'armeno, l'ebraico e l'arabo. La Tabella B.7, oltre ai tre esempi, mostra la struttura della codifica UTF-8, limitatamente a caratteri di uno o due byte (seconda e terza riga).

Bit carattere	Byte 1	Byte 2	Lettera	Cod. bin.	Cod. esa.
7	0bbbbbbb		a	0110 0001	0061
11	110bbbbb	10bbbbbb	à	1100 0011 1010 0000	C3A0
11	110bbbbb	10bbbbbb	δ	1100 1110 1011 0100	CEB4

Tabella B.7 Esempio di caratteri di uno (codice su 7 bit) o due byte (codice su 11 bit) secondo la codifica UTF-8. I bit "b" sono quelli che codificano il carattere. Per il formato di un byte la sequenza di "b" a destra dello 0 coincide con il codice ASCII.

B.8.3 BCD

La codifica BCD (*Binary Coded Decimal*) viene usata per rappresentare le cifre decimali su 4 bit come in Tabella B.8. Ovviamente la codifica BCD è meno compatta della codifica binaria. Ad esempio, il numero 147 in BCD è 0001 0100 0111; il medesimo numero in binario è 10010011.

La codifica BCD viene di norma usata nelle applicazioni di tipo commerciale, perché fornisce una conveniente rappresentazione dei numeri. Quasi tutte le macchine dispongono di aritmetica per la rappresentazione BCD.

Cifra Decimale	Codifica BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tabella B.8 Codifica BCD delle cifre decimali.

B.9 Unità aritmetiche e logiche

Al Paragrafo B.3 si è accennato all'aritmetica binaria. Si può fare un'osservazione, apparentemente ovvia ma di fondamentali conseguenze: le tabelline delle operazioni aritmetiche contengono i due simboli 0 e 1, esattamente come due sono i simboli nell'algebra delle reti. Ne deriva che le tabelline aritmetiche possono essere interpretate come le tabelle di verità delle funzioni logiche corrispondenti alle operazioni aritmetiche.

B.9.1 Semisommatore

Si consideri la somma di due bit. A sinistra di Figura B.9 è stata riportata la tabellina aritmetica della somma S e del relativo riporto R (Figura B.1). Qui appare evidente il vantaggio di usare per la logica i simboli 0 e 1 (non V/F, T/F, ecc.): la medesima tabellina può essere riguardata come la tabella di verità delle due funzioni logiche $S(A, B)$ e $R(A, B)$. Conseguentemente, se si costruisce la rete che ha come uscite S e R, si costruisce la rete che effettua la somma aritmetica di due bit e ne calcola il riporto. La rete in questione prende il nome di *Semisommatore* e viene indicata con HA (da *Half Adder*).

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

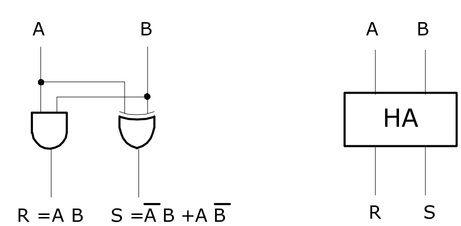


Figura B.9 Semisommatore. A sinistra si trova la tabellina aritmetica della somma e del riporto di due bit. La tabellina in questione viene reinterpretata come la tabella di verità delle due funzioni logiche S e R, dando luogo alla rete riportata al centro. A destra viene data una schematizzazione della rete come blocco funzionale. Il blocco è stato indicato come HA (da *Half Adder*), per *Semisommatore*.

B.9.2 Sommatore completo

Per poter effettuare la somma di due numeri interi su più bit occorre modificare il semisommatore appena visto in modo da tener conto dei riporti, passando al cosiddetto *Sommatore completo* (*Full Adder, FA*). Alla sinistra di Figura B.10 viene riportato lo schema del semisommatore con accanto la tabella di verità per la somma e il riporto in uscita, in funzione dei tre bit di ingresso A, B, R_i.

Per la realizzazione del sommatore completo si può seguire il seguente ragionamento.

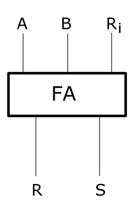
- La somma di tre bit A, B, R_i dà risultato 1 solo se è dispari il numero di bit a 1, cioè se è 1 l'OR esclusivo dei tre bit. In altri termini:

$$S = A \oplus B \oplus R_i = (A \oplus B) \oplus R_i$$

- Il riporto vale 1 quando: (a) la somma di A e B dà direttamente riporto; oppure (b) quando vale 1 la somma di A e B, con riporto in ingresso pure a 1. Questa descrizione a parole viene riformulata come:

$$R = AB + (A \oplus B)R_i$$

Dalle precedenti relazioni, tenuto conto che le due uscite del semisommatore rappresentano l'AND e lo XOR dei due ingressi, si ricava facilmente che il sommatore completo può essere costruito impiegando due semisommatori come nei due schemi di destra di Figura B.10.



A	B	R_i	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

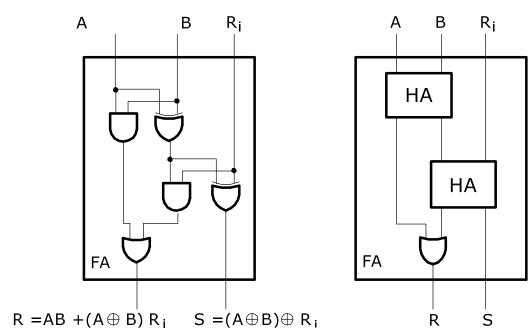


Figura B.10 A sinistra lo schema funzionale del sommatore completo con accanto la tabella di verità. Le due figure a destra mostrano sommatore completo costruito con due semisommatori.

B.9.3 Somma di due numeri interi

Il sommatore completo può essere impiegato in modo immediato per costruire un sommatore di interi di n bit, modellando il procedimento di somma con "carta e matita", come illustrato illustrato in Figura B.11, dove $A = [A_{n-1} \dots A_0]$ e $B = [B_{n-1} \dots B_0]$ sono i due termini della somma, $S = [S_{n-1} \dots S_0]$ il risultato, R_{n-1} il riporto. Ovviamente il riporto in ingresso al semisommatore del bit meno significativo è posto a 0.

Il tempo impiegato dalla rete di Figura B.11 per calcolare la somma dipende dalla lunghezza della parola e dalla particolare coppia di numeri sommati. Infatti la generica

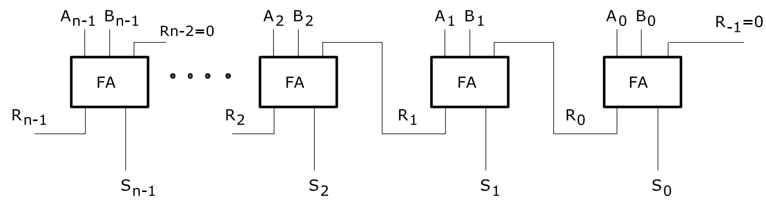


Figura B.11 Schema di un sommatore di parole di n bit, detto sommatore di *ripple*, costruito impiegando la cella detta *Sommatore completo* (FA).

cella produce una uscita stabile solo dopo che è diventato stabile il riporto in ingresso. Nel peggiore dei casi il riporto può propagarsi dal bit meno significativo a quello più significativo. Si indichino con τ_R e con τ_S i tempi di commutazione che la cella FA richiede per calcolare rispettivamente il riporto e la somma. Nel caso peggiore il riporto deve propagarsi attraverso tutte le celle. Si ha dunque un tempo di commutazione di caso peggiore pari a $\Delta_R = n\tau_R$, per il riporto R_{n-1}, e pari a $\Delta_S = (n-1)\tau_R + \tau_S$, per il calcolo della somma.

Se ora si assume che che tutte le porte commutino nello stesso tempo τ, i tempi richiesti da un FA per calcolare riporto e somma risultano pari a $\tau_R = 3\tau$ e $\tau_S = 2\tau$. Dunque, per un sommatore di n bit:

$$\Delta_S = (n-1)3\tau + 2\tau = (3n - 3 + 2)\tau = (3n-1)\tau$$

Calcolo anticipato del riporto

È possibile ridurre i tempi di calcolo della somma con la tecnica del calcolo anticipato del riporto. La tecnica si basa sulla struttura algebrica di S e R. Facendo riferimento alla Figura B.12e posti $p_i = A_i \oplus B_i$ e $g_i = A_iB_i$, si ha:

$$S_i = A_i \oplus B_i \oplus R_{i-1} = (A_i \oplus B_i) \oplus R_{i-1} = p_i \oplus R_{i-1}$$
$$R_i = A_iB_i + (A_i \oplus B_i)R_{i-1} = g_i + p_iR_{i-1}$$

Dunque:

$$R_0 = g_0 + p_0R_{-1}$$
$$R_1 = g_1 + p_1R_0 = g_1 + p_1g_0 + p_1p_0R_{-1}$$
$$R_2 = g_2 + p_2R_1 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0R_{-1}$$
$$R_3 = g_3 + p_3R_2 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0R_{-1}$$

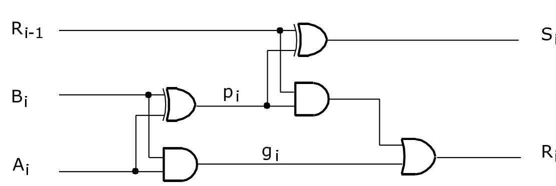


Figura B.12 Funzione generata (g_i) e funzione propagata (p_i) per il sommatore completo.

Nelle precedenti espressioni i termini p_i e g_i, detti rispettivamente *funzione propagata* e *funzione generata*, sono calcolati in un tempo τ pari alla commutazione di una sola porta. Gli R_i sono dunque calcolati in un tempo pari alla commutazione di tre porte. Di conseguenza, il calcolo di S richiede un tempo $\Delta_S = 4\tau$. La rete che effettua il calcolo dei riporti viene detta *look-ahead carry generator*. In Figura B.13 viene mostrato lo schema di un sommatore di parole di 4 bit con calcolo anticipato del riporto.

Considerando l'espressione di R_3 e posto: $G = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0$ e $P = p_3p_2p_1p_0$, R_3 si riscrive come: $R_3 = G + PR_{-1}$ È quindi possibile impiegare in modo iterativo il calcolo anticipato del riporto, costruendo reti a più livelli. Per esempio, si supponga di volere sommare parole di 16 bit. Il sommatore può essere costruito impiegando quattro sommatori da 4 bit e un ulteriore *look-ahead carry* generator. In pratica, sostituendo nella Figura B.13 i 4 FA da 1 bit con 4 FA da 4 bit.

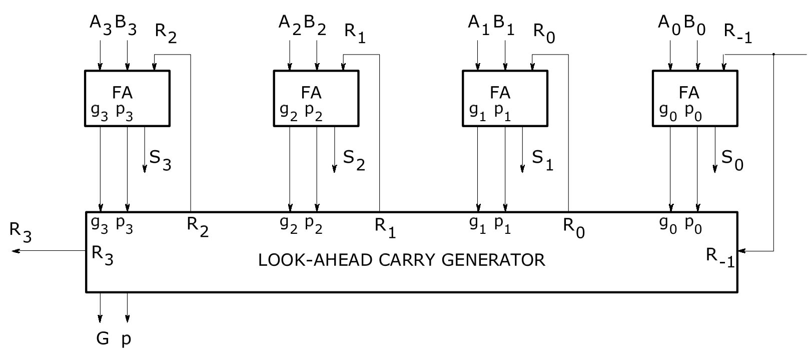


Figura B.13 Somma di parole di 4 bit con calcolo anticipato del riporto.

B.9.4 Esempio di costruzione di un'unità aritmetica

Ripartiamo dal sommatore completo di Figura B.10e vediamo come si costruisce una ALU, per ora di un solo bit. A tale scopo si faccia riferimento alla Figura B.14, dove attorno al blocco FA sono stati aggiunti due multiplexer, una porta AND, una OR e una NOT. Il multiplexer di destra attraverso l'ingresso di selezione[9] *op* (operazione) presenta in uscita una di queste alternative: (0) il risultato del calcolo di FA, (1) il risultato dell'AND, (2) il risultato dell'OR. Per quanto riguarda FA, l'ingresso di controllo *cb* (complementa B) determina cosa viene sommato: (0) somma $A + B + R_{in}$; (1) e $R_{in} = 1$ somma $A + \overline{B} + 1$ ovvero $A - B$. In sostanza la ALU di Figura B.14 è in grado sommare o sottrarre due bit, farne l'AND o l'OR.

Se ora vogliamo costruire una ALU da n bit, serviranno n ALU di un 1 bit. A titolo di esempio in Figura B.15 viene mostrata una ALU di 16 bit. Si noti che R_{in} del bit meno significativo è stato collegato a *cb*. In tal modo quando $cb = 1$ e $op = 0$ si comanda la ALU a effettuare la differenza $A - B$. In Figura B.15 è stata aggiunta una porta NOR avente come ingressi le linee di uscita della ALU. L'uscita della porta, indicata con Z dà 1 quando tutte le linee di ingresso sono a 0, ovvero quando è zero l'uscita

[9]Sul disegno si è indicato che l'ingresso di controllo *op* è costituito da 2 linee, in modo da codificare il numero di via tra 0, e 2.

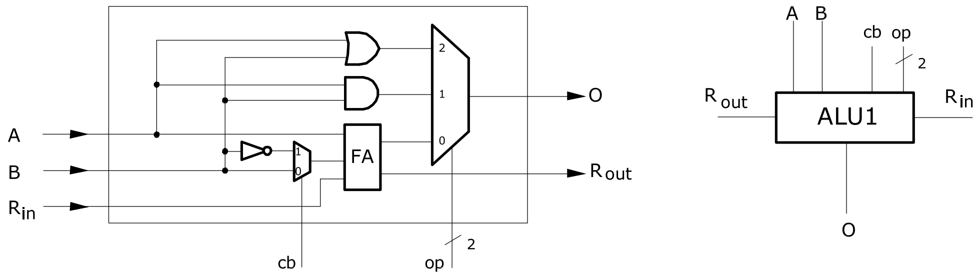


Figura B.14 Schema di una semplice ALU di 1 bit e relativa schematizzazione funzionale. Questa ALU è in grado di eseguire somma, sottrazione, AND e OR. L'ingresso di controllo cb seleziona $\overline{B}$ o B, mentre l'ingresso di controllo op (due linee) seleziona cosa esce dalla ALU ($op = 0$: uscita di FA, $op = 1$: A AND B, $op = 2$: A OR B).

$O = \{O_{15} \cdots O_0\}$ della ALU stessa. L'esecuzione dell'istruzione di salto condizionato `JZ` (salta se zero) si basa sullo stato della linea Z. Alternativamente si può immaginare di disporre dell'istruzione `JE` che confronta il contenuto di due registri (A e B), facendone la differenza, e salta se il risultato è zero, ovvero se $A = B$.

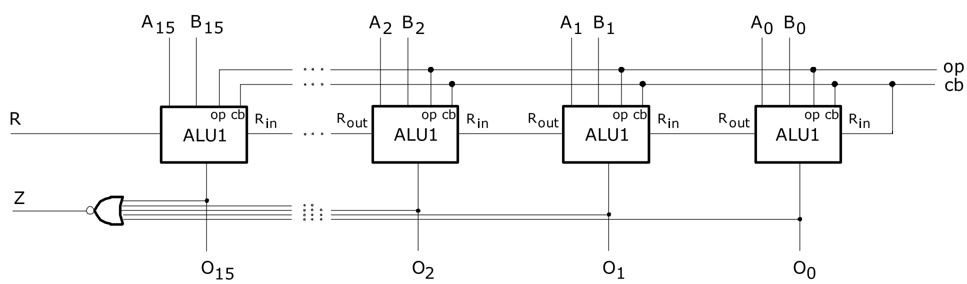


Figura B.15 Costruzione di una ALU a 16 bit dalla ALU a 1 bit.

La sola istruzione `JZ`, o l'equivalente `JE`, non è sufficiente a costruire una logica di programmazione completa. Occorre aggiungere almeno un'istruzione che faccia il confronto di maggiorità (`JG`) o di minorità (`JL`) tra due numeri. Simili istruzioni possono basarsi, come la precedente, sulla differenza. Tuttavia, la differenza, come pure la somma, possono portare a risultati inaspettati a causa del fenomeno del *trabocco* (*overflow*). Ad esempio, con parole di 8 bit si ha trabocco se la somma di due numeri positivi risulta maggiore di 127 (massimo numero rappresentabile su 7 bit), oppure se la somma di due numeri negativi risulta inferiore a -128.
Più precisamente, il rilievo della condizione di trabocco T si basa su queste considerazioni (ovviamente stiamo parlando di numeri in complemento a 2) :

a. La somma di due numeri di differente segno non può mai dare luogo a trabocco, mentre può esserci trabocco se il segno è uguale. C'è trabocco se sommando due numeri positivi si ottiene un risultato negativo, ovvero sommando due numeri negativi si ottiene un numero positivo.

b. La sottrazione di due numeri dello stesso segno non può mai dare luogo a trabocco. Nel sottrarre numeri di segno diverso c'è trabocco se il segno del risultato è differente da quello del minuendo (infatti, sottrarre da un minuendo un numero di segno differente è come sommare due numeri dello stesso segno del minuendo).

Tenuto conto che la sottrazione viene fatta con il medesimo FA, il trabocco si controlla verificando l'identità dei segni degli addendi e confrontandola con il segno risultante della somma, come in Figura B.16; ovvero

$$\begin{aligned}
T &= (A_{15} \cdot \overline{O}_{15} \cdot (B_{15} \cdot \overline{cb} + \overline{B}_{15} \cdot cb)) + (\overline{A}_{15} \cdot O_{15}(\overline{B_{15} \cdot \overline{cb} + \overline{B}_{15} \cdot cb})) \\
&= A_{15}\overline{O}_{15}B_{15}\overline{cb} + A_{15}\overline{O}_{15}\overline{B}_{15}cb + \overline{A}_{15}O_{15}\overline{B}_{15}\overline{cb} + \overline{A}_{15}O_{15}B_{15}cb \\
&= A_{15}B_{15}\overline{O}_{15}\overline{cb} + \overline{A}_{15}\overline{B}_{15}O_{15}\overline{cb} + A_{15}\overline{B}_{15}\overline{O}_{15}cb + \overline{A}_{15}B_{15}O_{15}cb \\
&= (A_{15}B_{15}\overline{O}_{15} + \overline{A}_{15}\overline{B}_{15}O_{15})\overline{cb} + (A_{15}\overline{B}_{15}\overline{O}_{15} + \overline{A}_{15}B_{15}O_{15})cb
\end{aligned}$$

Nella forma finale l'espressione conferma quanto detto ai punti a e b precedenti: il termine entro parentesi che moltiplica $\overline{cb}$ esprime quanto detto al punto a (somma); il termine entro parentesi che moltiplica cb esprime quanto detto al punto b (differenza).

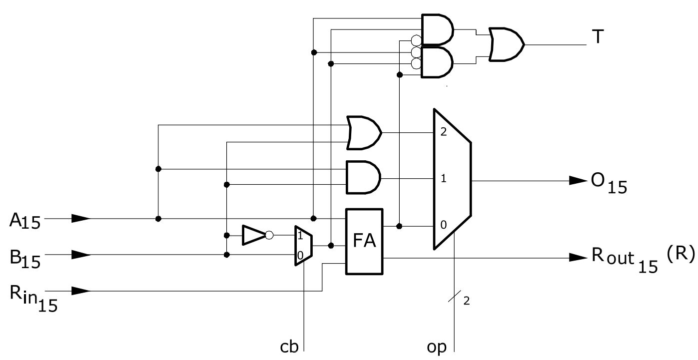


Figura B.16 Controllo del trabocco.

Vediamo ora cosa serve per l'istruzione `JG r1,r2,dest`, che fa saltare alla posizione simbolicamente indicata con `dest` se il contenuto del registro `r1` è maggiore del contenuto del registro `r2`. Assumiamo che i registri siano a 32 bit. Questi verranno presentati all'ALU per farne la differenza, `r1` come ingresso A, `r2` come ingresso B.
Si osserva che[10]:

- se i segni sono diversi si hanno due possibilità (non ci sarebbe nemmeno bisogno di fare la sottrazione):
  - A positivo (B negativo), ovvero è verificata la condizione ($\overline{A}_{31} \cdot B_{31} = 1$), allora $A > B$.
  - A negativo (B positivo), ovvero è verificata la condizione ($A_{31} \cdot \overline{B}_{31} = 1$), allora $A < B$.

[10]Si veda anche l'Esercizio B.31.

- se i segni sono uguali si hanno queste possibilità
  - segni positivi e risultato positivo ($\overline{A}_{31} \cdot \overline{B}_{31} \cdot \overline{O}_{31} = 1$), allora $A > B$;
  - segni positivi e risultato negativo ($\overline{A}_{31} \cdot \overline{B}_{31} \cdot O_{31} = 1$), allora $A < B$;
  - segni negativi e risultato negativo ($A_{31} \cdot B_{31} \cdot O_{31} = 1$), allora $A < B$;
  - segni negativi e risultato positivo ($A_{31} \cdot B_{31} \cdot \overline{O}_{31} = 1$), allora $A > B$;

Dunque si ha $A > B$ se è vera la relazione

$$\overline{A}_{31}B_{31} + \overline{A}_{31}\overline{B}_{31}\overline{O}_{31} + A_{31}B_{31}\overline{O}_{31} = \overline{A}_{31}B_{31} + \overline{A}_{31}\ \overline{O}_{31} + B_{31}\overline{O}_{31}$$

Si ha $A < B$ se è vera la relazione

$$A_{31}\overline{B}_{31} + \overline{A}_{31}\overline{B}_{31}O_{31} + A_{31}B_{31}O_{31} = A_{31}\overline{B}_{31} + A_{31}O_{31} + \overline{B}_{31}O_{31}$$

In Figura B.17 vengono mostrate le reti corrispondenti. È stata aggiunta anche la condizione di uguaglianza. Alle quattro uscite delle reti di figura si possono far corrispondere quattro istruzioni di salto condizionato (`JG`, `JGE`, `JL` e `JLE`).

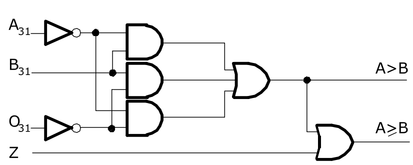


Figura B.17 Rilevamento delle condizioni di maggiorità e minorità

B.9.5 Moltiplicazione e divisione tra interi

Al Paragrafo B.3 sono state introdotte le operazioni su numeri binari interi positivi. Vogliamo ora illustrare come, a partire dall'unità di somma/sottrazione, possono essere realizzate le unità aritmetiche che effettuano la moltiplicazione e la divisione tra interi.

B.9.6 Moltiplicazione tra interi positivi

Con riferimento al metodo "carta e penna" esposto al paragrafo B.3, Figura B.3, 7, la moltiplicazione di due numeri di n bit può essere effettuata sommando n volte il moltiplicatore (moltiplicato per 0 o 1), facendolo scorrere verso sinistra dopo ogni somma parziale. Il corrispondente algoritmo viene mostrato in Figura B.18. Si deve assumere di utilizzare un sommatore di $2n$ bit per produrre un risultato (P) su $2n$ bit.

In Figura Figura B.19 viene mostrata la rete che svolge l'algoritmo. La rete è sequenziale: dopo l'inizializzazione di R, P e Q, ad ogni impulso di clock, il blocco CONTROL, asserisce i segnali di controllo LOAD, SHL e SHR nel modo che spieghiamo appoggiandoci alla Tabella B.9, dove, a titolo di esempio, vengono mostrati i passi della moltiplicazione di 1101(13) per 0111 (7). Con X_0 si è indicato il bit meno significativo di X. Su ogni rigo la tabella riporta l'effetto dell'operazione della seconda colonna, comandata da CONTROL. A parte l'inizializzazione, ogni passo da 1 a 4 corrisponde a un impulso di clock. Consideriamo, ad esempio il passo 1.

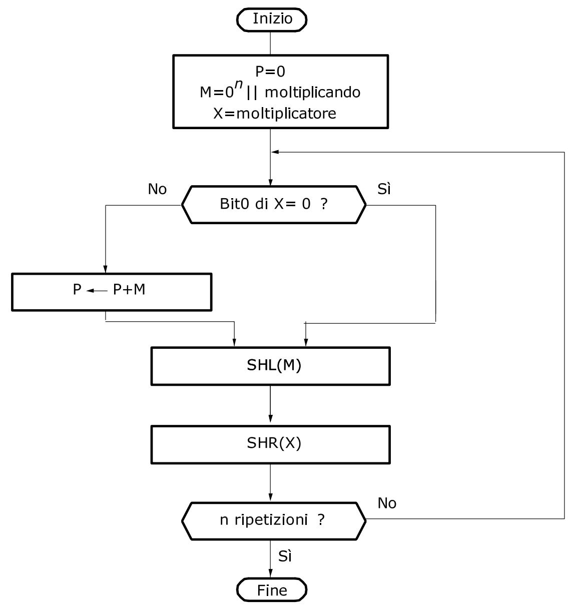


Figura B.18 Metodo "carta e penna" (Figura B.3, pag. 7) per la moltiplicazione di due numeri di n bit. X è il moltiplicatore di n bit; P ed M sono di 32 bit. Inizialmente P viene posto a zero, mentre M viene ottenuto concatenando il moltiplicando a n zeri (0^n||moltiplicando). Ad ogni scorrimento a sinistra di M (SHL(M)), nel suo bit meno significativo entra uno zero, mentre il bit più significativo (necessariamente uno zero) è perso. Al termine il risultato è in P.

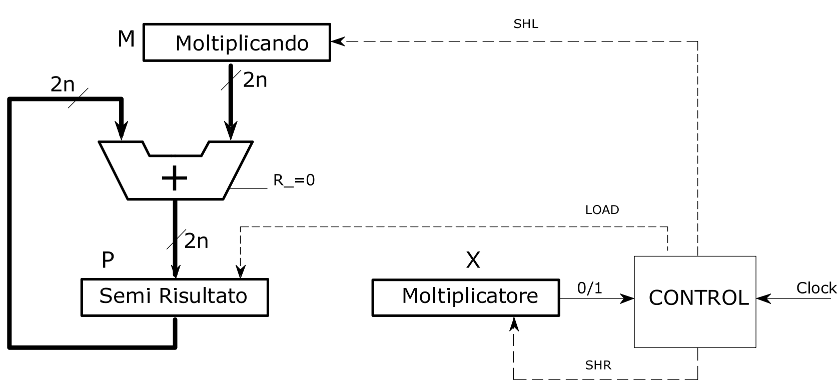


Figura B.19 Rete per la moltiplicazione di due numeri interi senza segno di n bit corrispondente all'algoritmo di Figura B.18. Le linee di controllo, generate dal blocco CONTROL sono state disegnate tratteggiate.

a) Poiché il bit meno significativo di X è 1 ($X_0 = 1$), viene asserito il comando LOAD. Ciò determina il caricamento in P dell'uscita del sommatore, ovvero di P+M. Nel caso specifico P $\leftarrow$ 00000000 + 00001101;
b) nella seconda fase del clock, viene ritirato il comando LOAD e vengono asseriti il comandi di scorrimento verso sinistra (SHL) per P e verso destra (SHR) per X.

Nel caso in cui il bit meno significativo di X non sia 1 il comando di LOAD non viene asserito; ciò equivale a sommare zeri a P. Dopo n clock il processo è concluso e il risultato è contenuto nel registro P. Nel caso specifico, dopo 4 iterazioni (pari al numero di bit del moltiplicatore), il risultato della moltiplicazione 0101 1011 (91) è in P.

Passo	Operazione	P	M	X
0	Inizializzazione	00000000	00001101	0111
1	X_0=1 $\Rightarrow$ LOAD $\Rightarrow$ P $\leftarrow$ P+M	00001101	00001101	0111
	SHL(M), SHR(X)	00001101	00011010	0011
2	X_0=1 $\Rightarrow$ LOAD $\Rightarrow$ P $\leftarrow$ P+M	00100111	00011010	0011
	SHL(M), SHR(X)	00100111	00110100	0001
3	X_0=1 $\Rightarrow$ LOAD $\Rightarrow$ P $\leftarrow$ P+M	01011011	00110100	0001
	SHL(M), SHR(X)	01011011	01101000	0000
4	X_0=0 $\Rightarrow$ —	01011011	01101000	0000
	SHL(M), SHR(X)	01011011	11010000	0000

Tabella B.9 Processo di esecuzione del prodotto di 1101 (13) con 0011 (7) con la rete di Figura B.19. La colonna di sinistra corrisponde al numero dei clock. Per ogni riga, sulle colonne P, M e X si legge lo stato dei registri dopo che è stata eseguita l'operazione della seconda colonna. Con X_0 si è indicato il bit meno significativo del moltiplicatore. Quando $X_0 = 1$ viene asserito LOAD e, conseguentemente, P viene caricato (P$\leftarrow$ P+M) con la somma dei contenuti di P ed M del rigo precedente. Quando $X_0 = 1$ non viene asserito LOAD, lasciando immutato P (equivale a sommare zeri). I segnali di controllo SHL e SHR vengono asseriti ad ogni passo. Al termine, il risultato 0101 1011 (91) si trova in P.

Miglioramento

L'algoritmo di Figura B.18 può essere migliorato, utilizzando un sommatore di n bit, tenendo fermo il moltiplicando, facendo scorrere verso destra i prodotti parziali e lo stesso moltiplicatore, e immettendo nel bit più significativo del moltiplicatore il bit che esce a destra da P. Ne consegue la rete di Figura B.20. Il registro M è usato per contenere permanentemente il moltiplicando; al termine dell'operazione, il registro P contiene la parte alta del risultato, il registro X la parte bassa; R è il bit di riporto; il riporto in ingresso al sommatore è tenuto permanentemente a 0. Inizialmente in M e X vengono rispettivamente caricati il moltiplicando e il moltiplicatore, mentre P e R vengono posti a zero (il caricamento di M e X, e l'azzeramento di P e R non sono mostrati in Figura B.20).

La rete risultante di Figura B.20 è sequenziale; dopo il caricamento iniziale, ad ogni impulso di clock, il blocco CONTROL, asserisce i segnali di controllo LOAD e SHR nel modo seguente:

1) nella prima fase del clock, se il bit meno significativo di X vale 1, viene asserito il comando LOAD, in modo che la somma dei contenuti di M e P venga depositata in P,

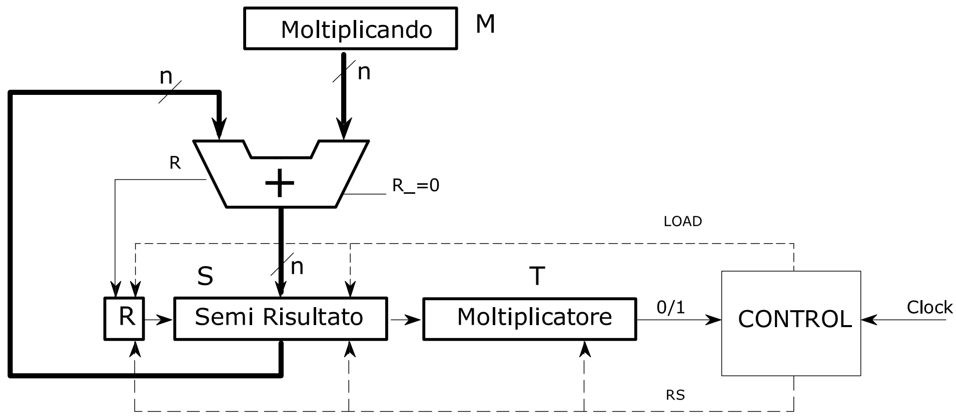


Figura B.20 Logica per la moltiplicazione tra numeri binari interi positivi di n bit.

e il riporto in R; se invece il bit meno significativo di X vale 0 il comando LOAD non viene asserito lasciando invariato il contenuto di P e R[11];

2) nella seconda fase del clock, viene asserito il comando di scorrimento verso destra (SHR), a seguito del quale (Figura B.20) il contenuto di R va nel bit più significativo di P, il bit meno significativo di P va nel bit più significativo di X, il bit meno significativo di X è perso; in R viene immesso 0 (per non complicare la Figura B.20 l'inserimento di 0 in R non è mostrato).

In altre parole, si effettuano le somme parziali e si sposta ogni volta verso destra il risultato. Il processo viene ripetuto per il numero di bit del moltiplicatore.

In Tabella B.10 si fa vedere il procedimento sempre nell'ipotesi di dover moltiplicare 1101 (13) per 0111 (7). Consideriamo ad esempio il passo 2. La condizione $X_0 = 1$ fa asserire LOAD e conseguentemente fa caricare in P la somma del contenuto di P (0110) con quello di M (1101). Dopo 4 iterazioni, il risultato 0101 1011 (91) è contenuto nella coppia di registri P-X.

La stessa rete di Figura B.20 funziona anche se il moltiplicando è negativo e il moltiplicatore negativo, salvo il fatto che il bit R deve essere sempre tenuto a 1, per infilare uni a sinistra, come in Figura B.5, pag. B.5 (si veda l'esercizio B.32).

Si noti che le reti delle Figure B.19 e B.20 sono sequenziali, ma, se le si immaginano nel contesto di un calcolatore, esse possono essere riguardate come reti combinatorie, la cui uscita è la somma dei due ingressi, salvo il fatto che l'operazione di moltiplicazione viene a richiedere un tempo pari almeno a $n \times T$, dove T è il periodo di clock.

[11] Alternativamente di può immaginare che il sommatore esegua sempre la somma, ma che sul suo ingresso di destra ci sia un multiplexer che, in questo caso scelga zero in luogo di M, in modo da mantenere invariato il contenuto di P.

Passo	Operazione	M	R	P	X
0	Inizializzazione	1101	0	0000	0111
1	X_0=1 ⇒ LOAD ⇒ P ← P+M	1101	0	1101	0111
	SHR(R||P||X)	1101	0	0110	1011
2	X_0=1 ⇒ LOAD ⇒ P ← P+M	1101	1	0011	1011
	SHR(R||P||X)	1101	0	1001	1101
3	X_0=1 ⇒ LOAD ⇒ P ← P+M	1101	1	0110	1101
	SHR(R||P||X)	1101	0	1011	0110
4	X_0=0 ⇒ —	1101	0	1011	0110
	SHR(R||P||X)	1101	0	0101	1011

Tabella B.10 Processo di esecuzione del prodotto di 1101 (13) con 0111 (7) con la rete di Figura B.20. Per quanto l'interpretazione delle operazioni valgono le indicazioni date per la Tabella B.9. Al termine il risultato 0101 1011 (91) si trova nella coppia di registri P-X.

B.9.7 Algoritmo di Booth

Basandoci su quanto detto al Paragrafo B.4.3, la rete di Figura B.20 può essere modificata come in Figura B.21, in modo da effettuare la moltiplicazione con il metodo di Booth. All'avvio, P viene azzerato, in M viene caricato il moltiplicando, in X il moltiplicatore, R e b vengono azzerati. Indicando con X_0 il bit meno significativo di X, il blocco CONTROL comanda la sottrazione e asserisce LOAD se (X_0 - b)= 1, comanda la somma e asserisce LOAD se (X_0 - b)= −1, non comanda la ALU né asserisce LOAD se (X_0 - b)= 0. Lo scorrimento verso destra viene eseguito ad ogni passo.

Il metodi di Booth dà risultati corretti anche nel caso di moltiplicatori negativi. Nella Tabella B.11 si fa vedere il prodotto 7×−3. Apparentemente sono gli stessi numeri usati nel caso della Tabella B.10, ma qui si assume che la rappresentazione sia in complemento a 2 su quattro bit, per cui la stringa 1101 è il numero −3. Sulla colonna di destra, l'azione da eseguire in base alla coppia (X_0,b) è indicata come somma o sottrazione tra P e M. Si deve intendere che il blocco CONTROL di Figura B.21 comanda la ALU all'operazione prevista e asserisce il LOAD verso P.

B.9.8 Divisione

Il procedimento di divisione impiegato al Paragrafo B.3, Figura B.4, corrisponde al metodo classico di divisione imparato sin dalle elementari. In sintesi:

1) si individua la porzione minima più a sinistra del dividendo tale che il suo contenuto sia pari o superiore al divisore; se ne fa la differenza ottenendo un resto parziale e assegnando 1 al quoziente;
2) si aggiungono tanti 0 in coda al quoziente quante sono le cifre del dividendo che devono essere "calate" e accodate al resto parziale per ottenere un numero pari o uguale a divisore; se ne fa la differenza ottenendo ancora un resto parziale e accodando 1 al quoziente;
3) si itera il passo 2) fino a che ci sono cifre del dividendo da considerare.

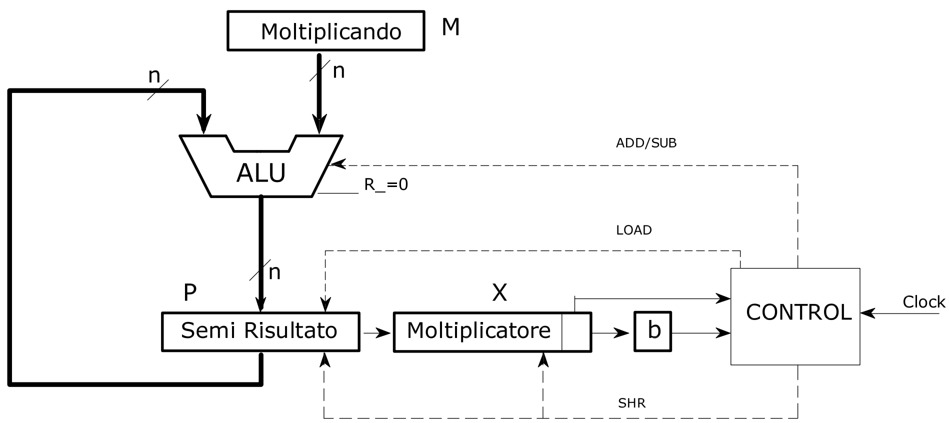


Figura B.21 Rete per la moltiplicazione col metodo di Booth. Con “b” si è indicato il bit con cui viene confrontato il bit meno significativo di X. In fase di inizializzazione b corrisponde al b_{-1} di pagina B.4.3 e pertanto viene posto a 0. Lo scorrimento verso destra di X porta il suo bit meno significativo in b (il contenuto di b è perso).

Passo	Operazione	M	P	X	b
0	Inizializzazione	0111	0000	1101	0
1	X_0b=10 ⇒ SUB, LOAD ⇒ P ← P−M	0111	1001	1101	0
	SHR(P||X||b)	0111	1100	1110	1
2	X_0b=01 ⇒ ADD, LOAD ⇒ P ← P+M	0111	0011	1110	1
	SHR(P||X||b)	0111	0001	1111	0
3	X_0b=10 ⇒ SUB, LOAD ⇒ P ← P−M	0111	1010	1111	0
	SHR(P||X||b)	0111	1101	0111	1
4	X_0b=11 ⇒ —	0111	1101	0111	1
	SHR(P||X||b)	0111	1110	1011	1

Tabella B.11 Processo di esecuzione del prodotto 7×−3 con la rete di Figura B.21 (si assume che la rappresentazione sia in complemento a 2 su 4 bit). Al termine il risultato 1110 1011 (−21) si trova nella coppia di registri P-X.

L’algoritmo sopra descritto si traduce nella rete illustrata in Figura B.22. La linea Segno, in uscita dal sottrattore, indica se il sottraendo è minore, ovvero maggiore-uguale, del divisore.

Inizialmente il dividendo e il divisore vengono caricati rispettivamente in Q e D, mentre il registro R è posto a zero. Lo scorrimento verso sinistra del registro Q porta a identificare la porzione più significativa del dividendo cui viene sottratto il divisore e successivamente ad aggiungere le restanti cifre del dividendo in coda ai resti parziali in modo da formare termini maggiori o uguali al divisore. Nello scorrimento verso sinistra si aggiunge 0 in coda a Q quando non si effettua la sottrazione, 1 quando la sottrazione ha luogo. Il procedimento ha termine quando tutti i bit del dividendo sono stati considerati,

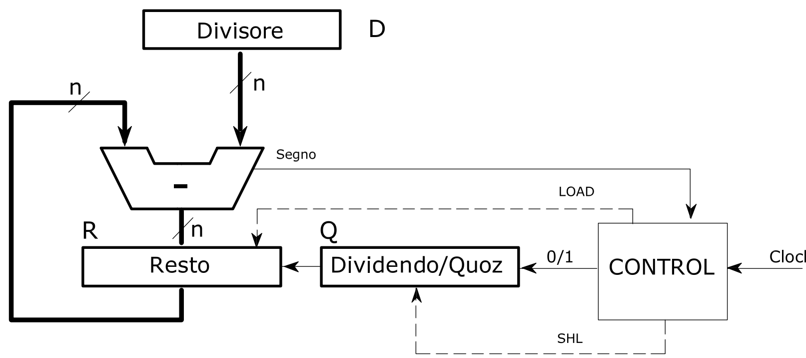


Figura B.22 Logica per la divisione secondo l'algoritmo di Figura B.23

Passo	Operazione	D	R	Q
0	Inizializzazione	00101	00000	10001
1	SHL(R||Q)	00101	00001	0001–
	R<D ⇒ Q←0	00101	00001	00010
2	SHL(R||Q)	00101	00010	0010–
	R<D ⇒ Q←0	00101	00010	00100
3	SHL(R||Q)	00101	00100	0100–
	R<D ⇒ Q←0	00101	00100	01000
4	SHL(R||Q)	00101	01000	1000–
	R>D ⇒ Q←1, LOAD ⇒ R← R−D	00101	00011	10001
5	SHL(R||Q)	00101	00111	0000–
	R>D ⇒ Q←1, LOAD ⇒ R← R−D	00101	00010	00011

Tabella B.12 Processo di divisione di 10001 (17) con 101 (5). Al termine il resto 10 (2) è in R e il quoziente 11 (3) è in Q.

ovvero quando tutti i bit inizialmente caricati in Q sono stati spostati in R.

Più precisamente, dopo l'inizializzazione dei registri, la divisione avviene iterando n volte i seguenti due passi:

1) viene effettuato lo scorrimento a sinistra di R e Q. Il segno generato dal sottrattore determina l'azione successiva;
2) se il segno è negativo (R<D), allora nel bit meno significativo di Q viene inserito 0; in caso contrario (R≥D) nel bit meno significativo di Q viene inserito 1 e viene asserito il comando di LOAD in modo da caricare in R la differenza R−D.

I passi del procedimento vengono illustrati in Tabella B.13 con riferimento alla divisione di 10001 (17) per 101 (5). Per i primi 3 passi Q viene fatto scorrere e viene inserito 0; al quarto passo, in base alla condizione R>D, viene inserito 1 Q e asserito LOAD che determina R←R−D (01000 − 00101 = 00011).

Il precedente algoritmo ha un difetto: viene prima fatto lo scorrimento, quindi valutato il segno e, all'ultimo, inserito il bit dovuto nell'ultima posizione di Q (la rete di Figura B.22 non mostra il dettaglio di quest'ultimo aspetto). Si osservi però che, occorre comunque almeno uno scorrimento prima di poter individuare la porzione più significativa del dividendo. Dunque l'algoritmo può essere migliorato prevedendo in fase di inizializzazione il primo scorrimento di Q con l'inserzione di 0. In tal caso il procedimento richiede $n - 1$ iterazioni, in ciascuna delle quali l'inserimento del bit a destra di Q avviene in contemporanea allo scorrimento. L'algoritmo risultante è illustrato nel diagramma di Figura B.23. La rete di Figura B.22 è conforme a questo algoritmo e l'esempio precedente viene eseguito come in Tabella B.12.

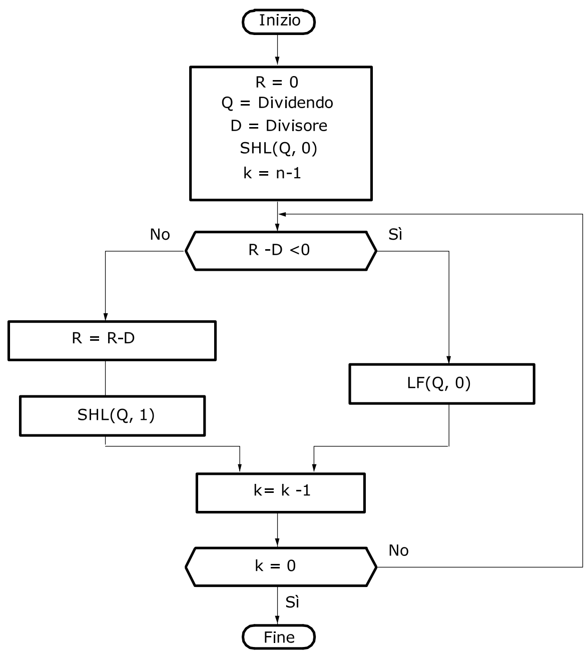


Figura B.23 Algoritmo di divisione. SHL(Q,0) e SHL(Q,1) stanno per lo scorrimento a sinistra di Q con l'inserimento di 0 e 1 rispettivamente.

B.10 Siti Web

Il sito `http://www.unicode.org/` spiega la codifica UNICODE.
Il sito `https://it.wikipedia.org/wiki/UTF-8` illustra l'UTF-8 e presenta molti rinvii ad altre pagine, tra cui quelle che danno le codifiche dei caratteri.

Passo	Operazione	D	R	Q
0	Inizializzazione	00101	00001	00010
1	R<D ⇒ SHL(R||Q||0)	00101	00010	00100
2	R<D ⇒ SHL(R||Q||0)	00101	00100	01000
3	R<D ⇒ SHL(R||Q||0)	00101	01000	10000
4	R<D ⇒ SHL(R||Q||1), LOAD ⇒ R← R-Q	00101	00111	00001
5	R<D ⇒ SHL(R||Q||1), LOAD ⇒ R← R-Q	00101	00010	00011

Tabella B.13 Procedimento di divisione di 10001 (17) con 101 (5) secondo l'algoritmo di Figura B.23. Al termine il resto 10 (2) è in R e il quoziente 11 (3) è in Q.

Domande ed esercizi

B.1 Le dita della mano hanno 3 falangi. In alcuni paesi si usa contare in base 12 (prodotto di 3 con il numero delle dita, escluso il pollice). Si provi a sommare due numeri tenendo traccia della somma corrente sulle falangi. Sa dire il lettore qual è il vantaggio rispetto alla numerazione in base 10?

B.2 Si discuta la ragione per la quale, quando si parla del contenuto di una parola di memoria o di un registro di un calcolatore si usa la rappresentazione in base 16.

B.3 In passato, nel mondo dei calcolatori, è stata a lungo usata la rappresentazione in base 8 (2^3). Si individuino i motivi per i quali la notazione in base 16 è da preferirsi a quella in base 8. In particolare si esamini il problema della rappresentazione del contenuto di byte contigui (un byte è di 8 bit e quindi non è scomponibile in gruppi di 3 bit).

B.4 Convertire in binario i seguenti numeri in base 10:

(a) 143	(b) 312	(c) 91	(d) 123
(e) 0,7155	(f) 0,312	(g) 0,72	(h) 0,345
(i) 7,55	(l) 12,5	(m) 3,12	(n) 11,77

B.5 Convertire in base 10 i seguenti numeri in base 2:

(a) 1011	(b) 1000111	(c) 11111000	(d) 0,11101
(e) 0,0011111	(f) 11,101	(g) 10101,10101	(h) 11,000001

B.6 Convertire in notazione decimale i seguenti numeri esadecimali:

(a) 1AB0	(b) ABCD	(c) 0,123	(d) AB,12D

B.7 Convertire in notazione decimale i seguenti numeri ottali:

(a) 76022	(b) 1010	(c) 0,66	(d) 23,1010

B.8 Convertire i seguenti numeri decimali in notazione esadecimale e ottale:

(a) 3500	(b) 531	(c) 0,12345	(d) 35,7

B.9 Convertire i seguenti numeri dalla base 16 alle basi 8, 4 e 2.

(a) D15	(b) 64	(c) ABCD	(d) FFE

Convertire i seguenti numeri dalla base 8 alle basi 16, 4 e 2.

(a) 67201	(b) 10777	(c) 73601	(d) 64

B.10 Convertire i seguenti numeri:

(a) 201102 da base 3 a base 16	(b) 3201 da base 4 a base 7
(c) 303 da base 4 a base 6	(d) 754 da base 9 a base 16.

B.11 Convertire i seguenti numeri dalla base 9 alla base 3.

(a) 82704	(b) 64	(c) 108887	(d) 12345

Convertire i seguenti numeri dalla base 3 alla base 9.

(a) 211200212	(b) 21022	(c) 20001	(d) 202101

B.12 Si effettuino le seguenti operazioni binarie su numeri senza segno:

(a) $1100101 + 1001101$ (b) $11 + 100$ (c) $10110 - 00111$ (d) $111 - 011$
(e) 1100×1001 (f) $1011 \times 0,100$ (g) $10110 : 101$ (h) $111,1011 : 0,11$

B.13 Sulle auto italiane il numero di targa è su 7 cifre, di cui le prime 2 e le ultime 2 sono prese dalle lettere dell'alfabeto, mentre le 3 centrali sono cifre decimali.
a) si calcoli qual è il corrispondente massimo numero decimale;
b) si calcoli a quale numero decimale corrisponde la targa EG401YF.

B.14 Si effettuino direttamente nella rispettiva aritmetica le seguenti operazioni tra numeri in base diversa da quella decimale:

(a) $202_3 + 101_3$ (b) $257_8 - 167_8$ (c) $412_5 \times 101_5$
(d) $5420_6 : 4_6$ (e) $1CFDF_{16} + 89_{16}$ (f) $257_8 - 167_8$
(g) $A0C6_{16} - F1A_{16}$ (h) $243_8 - 14_8$ (i) $66_8 : 10_8$

B.15 Si effettui la differenza tra il numero esadecimale A0E3 e il numero ottale 6755. La differenza va effettuata o in notazione esadecimale o in notazione ottale. Si riporti il risultato a notazione decimale e si verifichi che è corretto.

B.16 Effettuare le seguenti sottrazioni usando la rappresentazione in complemento a 2 (i numeri dati sono da intendere come interi senza segno):

(a) $11100 - 10100$ (b) $1100 - 101$ (c) $0,101 - 0,011$ (d) $101,11 - 100,101$

B.17 Si convertano in forma binaria i numeri 23 e 6 (rappresentandoli per brevità su 6 bit), si effettuino queste moltiplicazioni 23×-6 e -23×-6. Si verifichi che il metodo "carta e matita" dà risultati errati essendo il moltiplicatore negativo (si veda anche l'Esercizio B.18).

B.18 Si convertano in forma binaria i numeri 23 e 6 e si effettuino le moltiplicazioni seguenti con l'algoritmo di Booth; si verifichi che il metodo opera indifferentemente con numeri positivi e negativi in complemento a 2

(a) 23×6 (b) -23×6 (c) 23×-6 (d) -23×-6

B.19 Prima dell'affermazione dello standard IEEE sull'aritmetica in virgola mobile, ogni costruttore usava le sua convenzioni. Si approfondiscano i vantaggi che derivano da una rappresentazione standardizzata.

B.20 Discutere la ragione per cui la costante di polarizzazione dello standard IEEE-745 è pari a 127 e non 128, come sarebbe stato possibile con un campo esponente di 8 bit.

B.21 Qual è il vantaggio di avere come bit nascosto il primo 1 e non 0?

B.22 Si trasformino i seguenti numeri in formato IEEE singola precisione, dando il valore di mantissa, segno ed esponente:

(a) 14,3 (b) 3,14 (c) 12,34 (d) −1234567,6631

B.23 La codifica ASCII dei caratteri alfanumerici, ai giorni nostri risulta alquanto limitata in quanto non prevede la rappresentazione di caratteri come quelli che, ad esempio, hanno la dieresi o la cediglia. Il lettore è invitato a individuare e a confrontare con lo standard ASCII la codifica dei caratteri in uso sul suo PC.

B.24 Il contenuto di un byte è 0110 1001. Che cosa rappresenta tale stringa di bit se la si interpreta come numero binario o codice ASCII?

B.25 Il contenuto di tre parole di memoria di 16 bit consecutive in forma esadecimale è 4C31, 6A79, 3337. A quale stringa di caratteri ASCII corrispondono?

B.26 Data una stringa di caratteri ASCII, per esempio “2836”, si definisca un algoritmo per calcolare il corrispondente binario. La stringa è rappresentata da byte dislocati in posizioni successive di memoria.

B.27 Si considerino i due numeri 79 e 48. Si rappresentino in BCD e si esegua la somma sulla rappresentazione BCD. Si definisca la tabella per la somma in BCD tenendo conto del riporto.

B.28 In riferimento al semi-sommatore se ne dia una realizzazione con sole porte NOR e una con sole porte NAND.

B.29 Si calcoli il tempo di commutazione Δ_S di un sommatore costruito con il sommatore completo di Figura B.10.

B.30 Usare il sommatore di 4 bit di Figura B.13 per costruire un sommatore di 16 bit. Calcolare il tempo di commutazione su S_{15} (Δ_S) e su R_{16} (Δ_R).

B.31 Si verifichino le relazioni circa le condizioni di maggiorità/minorità del Paragrafo B.9.4, con riferimento alla differenza. A tale scopo di prendano due numeri 7 e 5, rappresentati su 4 bit e si provi che, per tutte le combinazioni dei segni, valgono le relazioni dette.

B.32 Verificare se la rete di Figura B.20 effettua correttamente la moltiplicazione di un numero negativo per uno positivo. A tal fine si assuma che i due numeri dati 1101 e 0111 siano da interpretare con segno, ovvero come -3 e 7.

B.33 Si verifichi che la rete di Figura B.21 effettua correttamente la moltiplicazione secondo l'algoritmo di Booth, costruendo la tabella del processo similmente alle tabelle B.9 e B.10.

B.34 La rete di Figura B.24 è lo schema di principio di un sommatore seriale. RA, RB e RC sono registri a scorrimento, FA è un full adder. RA e RB vengono caricati con i due numeri da sommare e R azzerato. Ad ogni impulso di clock viene effettuata la somma dei due bit a destra di RA e RB con R. Si esamini il funzionamento della rete tenendo traccia, clock dopo clock, del contenuto dei registri e di R. Si supponga che ad ogni scorrimento in RA e RB entri uno 0. Si consideri il caso della somma di due numeri in complemento a 2, rappresentati su 5 bit.

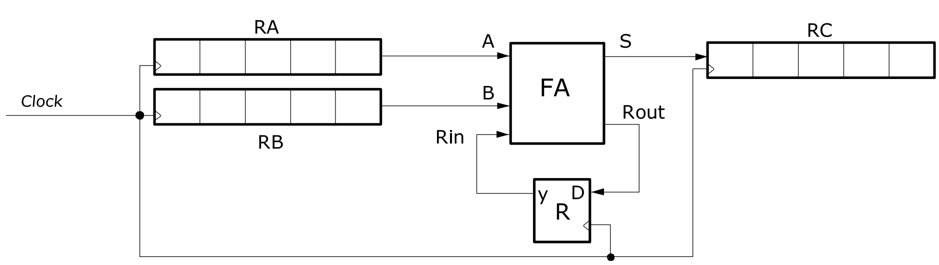


Figura B.24 Rete per l'Esercizio B.34.

Soluzioni degli esercizi dell'Appendice B

Aggiornato il 31 marzo 2017

B.2 La rappresentazione esadecimale prevede 16 configurazioni corrispondenti a 4 bit. Il contenuto di una parola di 16 bit può essere rappresentato direttamente con 4 digit esadecimali, sostituendo a ogni gruppo di 4 bit (*nibble*) la corrispondente cifra esadecimale.

B.3 A titolo di esempio si consideri una parola di 16 bit contenente 0011 1011 1001 0111. In notazione esadecimale essa si rappresenta come 3B97, ovvero come due byte contenenti 3B e 97.

In notazione ottale essa si rappresenta come 035627; ma se vogliamo rappresentare i due byte essi sarebbero 073 (il più significativo) e 227 (il meno significativo). Con la rappresentazione ottale si dà una lettura diversa per i contenuti dei byte a seconda del fatto che siano considerati in modo indipendente o come parte di una parola.

B.4 La conversione da base decimale a binaria di un numero intero si ottiene per successive divisioni per 2 (Paragrafo B.2.1, pag. B.2.1). La successione dei resti rappresenta, dalla cifra meno significativa alla più significativa, il numero binario.

Mostriamo il processo di conversione in riferimento al numero 143_{10}. La successione di quozienti e resti ottenuti dividendo per 2 è:

(71,1), (35,1), (17,1), (8,1), (4,0), (2,0), (1,0), (0,1)

dunque: $143_{10} = 10001111_2$.

Applicando lo stesso procedimento si ottiene: $312_{10} = 100111000_2$; $91_{10} = 1011011_2$

Per quanto si riferisce ai numeri frazionari si tratta di applicare la tecnica del Paragrafo B.2.1. Considerando, ad esempio, il numero 0,7155 e limitandoci alle prime otto cifre significative si ha:

$$\begin{aligned}
0,7155 \times 2 &= 1,431 \\
0,431 \times 2 &= 0,862 \\
0,862 \times 2 &= 1,724 \\
0,724 \times 2 &= 1,448 \\
0,448 \times 2 &= 0,896 \\
0,896 \times 2 &= 1,792 \\
0,792 \times 2 &= 1,584 \\
0,584 \times 2 &= 1,168
\end{aligned}$$

Dunque $0,7155_{10} = 0,10110111_2$

Per i numeri in virgola si applicano ambedue le regole. Considerando il numero 11,77, relativamente a 11 si ha questa successione di quozienti e resti ottenuti dividendo per 2:

(5,1), (2,1), (1,0), (0,1)

dunque: $11_{10} = 1011_2$ mentre per la parte frazionaria (fermandosi alla sesta cifra decimale) si ha:

$$\begin{array}{rcl} 0,77 \times 2 & = & 1,54 \\ 0,54 \times 2 & = & 1,08 \\ 0,08 \times 2 & = & 0,16 \\ 0,16 \times 2 & = & 0,32 \\ 0,32 \times 2 & = & 0,64 \\ 0,64 \times 2 & = & 1,28 \end{array} \tag{B.2}$$

dunque $0,77_{10} = 0,110001_2$. In conclusione $11,77 = 1011,110001_2$.

B.5 Per la conversione da binario a decimale si calcola il polinomio delle potenze di 2 i cui coefficienti sono dati dalla stringa binaria di partenza (Paragrafo B.2.1). Pertanto al numero (a) 1011_2 corrisponde il polinomio:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11_{10}$$

In modo del tutto analogo si ottiene: $1000111_2 = 71_{10}$ e $1010001_2 = 81_{10}$. Per il numero (f) $11,101_2 = 1\times 2^1 + 1\times 2^0, 1\times 2^{-1} + 0\times 2^{-2} 1\times 2^{-3} = 2+1, 1/2+1/8 = 3, 0,5+0,125 = 3,625_{10}$.

B.6 La conversione da una generica base alla base decimale si ottiene col metodo del calcolo del polinomio di potenze della base di partenza.

Al numero $1AB0_{16}$ corrisponde il polinomio:

$$1 \times 16^3 + A \times 16^2 + B \times 16^1 + 0 \times 16^0 = 1 \times 16^3 + 10 \times 16^2 + 11 \times 16^1 + 0 \times 16^0 = 6832_{10}$$

B.7 Per la conversione del numero 76022_8 si ha:

$$7 \times 8^4 + 6 \times 8^3 + 0 \times 8^2 + 2 \times 8^1 + 2 \times 8^0 = 31762_{10}$$

B.8 La conversione dalla base decimale ad una base generica si ottiene per successive divisioni del numero da convertire per la base di arrivo. Per il numero 3500_{10} si ottiene:

$$\begin{array}{lll} 3500: & \underline{16} & \\ 12 & 218: & \underline{16} \\ & 10 & 13 \end{array}$$

Tenuto conto che $12_{10} = \mathrm{C}_{16},\ 10_{10} = \mathrm{A}_{16},\ 13_{10} = \mathrm{D}_{16}$, ne consegue:

$$3500_{10} = \mathrm{DAC}_{16}$$

La conversione a base 8 si ottiene in modo del tutto analogo:

$$\begin{array}{llll}
3500:\underline{\ 8\ } & & & \\
4 \quad 437:\underline{\ 8\ } & & & \\
\quad 5 \quad 54:\underline{\ 8\ } & & & \\
\qquad\quad 6 \quad 6 & & &
\end{array}$$

Dunque $3500_{10} = 6654_8$.

Analogamente si ottiene: $531_{10} = 213_{16}$; $531_{10} = 1023_8$

B.9 La conversione tra basi che sono potenze l'una dell'altra si ottiene raggruppando opportunamente le cifre del numero espresso nella base più piccola (Paragrafo B.2.2). Dunque basta convertire prima il numero dato nella base più piccola e poi effettuare le altre conversioni. Ad esempio:

$$\begin{array}{rrrr}
D15_{16} = & 1101\ 0001\ 0101_2 = & 110\ 100\ 010\ 101 = & 6425_8 \\
 & = & 1\ 10\ 11\ 10\ 10\ 00\ 00\ 01 = & 310111_4
\end{array}$$

In maniera analoga:

$$\begin{array}{rrrr}
67201_8 = & 110\ 111\ 010\ 000\ 001_2 = & 1\ 10\ 11\ 10\ 10\ 00\ 00\ 01 = & 12322001_4 \\
 & = & 110\ 1110\ 1000\ 0001 = & 6E81_{16}
\end{array}$$

B.10 Per la conversione tra basi che non comprendono la base decimale né siano tra loro potenze l'una dell'altra, conviene passare dalla base di partenza alla base decimale calcolando il polinomio corrispondente e dalla base decimale alla base di arrivo utilizzando le successive divisioni.

Per la converversione di 201102 da base 3 a base 16 si ha:

$$201102_3 = 2 \times 3^5 + 0 \times 3^4 + 1 \times 3^3 + 1 \times 3^2 + 0 \times 3^1 + 2 \times 3^0 = 524_{10}$$

Segue:

$$\begin{array}{l}
524:\underline{\ 16\ } \\
C \quad 32:\underline{\ 16\ } \\
\qquad\ 0 \quad 2
\end{array}$$

In conclusione: $201102_3 = 20C_{16}$

Per la conversione di 3201 da base 4 a base 7 si ha:

$$3 \times 4^3 + 2 \times 4^2 + 0 \times 4^1 + 1 * 4^0 = (225)_{10}$$

e quindi:

$$\begin{array}{l}
225:\underline{\ 7\ } \\
1 \quad 32:\underline{\ 7\ } \\
\qquad 4 \quad 4
\end{array}$$

Dunque:

$$3201_4 = (441)_7$$

Per le altre conversioni si ha: $303_4 = 123_6$; $754_9 = 268_{16}$

B.11 Il numero 9 è una potenza del 3; dunque il passaggio tra le due basi si ottiene raggruppando opportunamente le cifre (come nel passaggio tra la base 2 e la base 4). Ci si può aiutare facendo riferimento alla TabellamB.1.

$$82704_9 = 8\ 2\ 7\ 0\ 4_9 = 22\ 02\ 21\ 00\ 11_3 = 2202210011_3$$

$$211200212_3 = 2\ 11\ 20\ 02\ 12_3 = 2\ 4\ 6\ 2\ 5_9 = 24625_9$$

Nel tradurre da una base all'altra si deve tener conto, per esempio, che 4_9 si trasforma in 11_3, 8_9 si trasforma in 22_3, ovvero 12_3 $(= 5_{10})$ si trasforma in 5_9.

B.12 Vengono risolti alcuni esempi (si indica il riporto solo se è 1):
Caso (a)

$$\begin{array}{cccccccccl}
 & & & 1 & 1 & & 1 & & & \text{Riporti} \\
 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & + & \text{Primo addendo} \\
 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & = & \text{Secondo addendo} \\ \hline
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & & \text{Risultato}
\end{array}$$

Dunque $1100101_2 + 1001101_2 = 10110010_2$
In modo analogo per il caso (b): $011_2 + 100_2 = 111_2$
Caso (c), sottrazione (si indica il prestito solo se è 1):

$$\begin{array}{cccccccl}
 & 1 & 1 & 1 & & & & \text{Prestiti} \\
1 & 0 & 1 & 1 & 0 & - & & \text{Minuendo} \\
0 & 0 & 1 & 1 & 1 & = & & \text{Sottraendo} \\ \hline
0 & 1 & 1 & 1 & 1 & & & \text{Risultato}
\end{array}$$

Dunque $10110_2 - 00111_2 = 01111_2$
Analogamente, per il caso (d): $111_2 - 011_2 = 100_2$
Per il caso (g): $(22/5 = 4$ con resto $2)$

$$\begin{array}{cccccc|ccc}
1 & 0 & 1 & 1 & 0 & : & 1 & 0 & 1 \\ \cline{7-9}
1 & 0 & 1 & & & & 1 & 0 & 0 \\ \cline{1-3}
0 & 0 & 0 & 1 & 0 & & & &
\end{array}$$

Ovvero, il quoziente è 100 (4) e il resto è 10 (2).

B.14 L'operazione di somma si esegue cifra per cifra come se fosse in base dieci; al risultato si toglie il numero degli elementi della base in cui sono espressi i numeri da sommare. Ciò che si ottiene è un risultato parziale della somma richiesta. Si procede così fino al termine tenendo conto dei riporti delle somme parziali precedenti.

Caso (a) (202+101 in base 3):

$$\begin{array}{cccccl}
 & & 1 & & & \text{Riporti} \\
 & 2 & 0 & 2 & + & \text{Primo addendo} \\
 & 1 & 0 & 1 & = & \text{Secondo addendo} \\ \hline
1 & 0 & 1 & 0 & & \text{Risultato}
\end{array}$$

Caso (b) (257-167 in base 8):

$$\begin{array}{cccc l}
1 & & & & \text{Prestiti} \\
2 & 5 & 7 & - & \text{Minuendo} \\
1 & 6 & 7 & = & \text{Sottraendo} \\
\hline
0 & 7 & 0 & & \text{Risultato}
\end{array}$$

Naturalmente il modo meno soggetto a errore per eseguire operazioni aritmetiche fra numeri espressi in una base diversa da quella decimale, si convertono i numeri nella base dieci, si esegue l'operazione in questa base e si converte poi il risultato nella base di partenza.

Caso (c):

Essendo $412_5 = 107_{10}$ e $101_5 = 26_{10}$ ed essendo $107_{10} \times 26_{10} = 2782_{10}$; poichè $2782_{10} = 42112_5$ ne consegue $412_5 \times 101_5 = 42112_5$

Caso (d):

$5420_6 = 1236_{10}$ e $4_6 = 4_{10}$; $1236_{10} : 4_{10} = 309_{10}$; poichè $309_{10} = 1233_6$ ne consegue $5420_6 : 4_6 = 1233_6$

B.15 Scegliamo di effettuare l'operazione richiesta in base ottale, per cui, si ha:

$$A0E3_{16} = 1010000011100011_2 = 120343_8$$

ovvero

$$120343_8 - 6755_8 = 111366_8 = 37622_{10}$$

Verifica:

$$A0E3_{16} = A \times 16^3 + 0 \times 16^2 + E \times 16^1 + 3 \times 16^0 = 41187_{10}$$

$$6755_8 = 6 \times 8^3 + 7 \times 8^2 + 5 \times 8^1 + 5 \times 8^0 = 3565_{10}$$

Quindi $41187 - 3565 = 37622$

B.17 Rappresentando i numeri binari su 6 bit, il numero 23 è pari a 01 0111, quindi -23 è 101001; 6 è pari a 000110, quindi -6 è 111010. I due prodotti sono $23 \times -6 = -138$ (1111 0111 0110) e $-23 \times -6 = 138$ (0000 1000 1010)

$23 \times -6 = 138$

```
      0 1 0 1 1 1 × 1 1 1 0 1 0
      0 0 0 0 0 0
    0 1 0 1 1 1
  0 0 0 0 0 0
0 1 0 1 1 1
0 1 0 1 1 1
0 1 0 1 1 1
0 1 0 1 0 0 1 1 0 1 1 0
```

-23×-6

```
            1 0 1 0 0 1 × 1 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 0 1
1 1 1 0 1 0 0 1
1 1 0 1 0 0 1
1 1 1 0 1 1 0 0 1 0 1 0
```

È subito evidente che nessuna delle due dà risultato corretto. Si veda anche l'Esercizio B.18

B.18 Le conversioni danno luogo ai seguenti numeri in complemento a 2, dove i termini del prodotto sono su 6 bit e il risultato su 12.
6: 00 0110; -23 : 101001; -6 : 111010;
138: 0000 1000 1010; -138 : 111101110110.
Il moltiplicatore di Booth è "$0+10-10$" per 0110 e "$0-1+1-10$" per 11010.

$23 \times 6 = 138$

Moltiplicando					0	1	0	1	1	1	×
Moltiplicatore di Booth							+1	0	−1	0	
	0	0	0	0	0	0	0	0	0	0	
Compl. a 2 del Moltiplicando→	1	1	1	1	0	1	0	0	1		
	0	0	0	0	0	0	0	0			
	0	0	1	0	1	1	1				
Prodotto	0	0	1	0	0	0	1	0	1	0	

$23 \times -6 = -138$

Moltiplicando					0	1	0	1	1	1	×
Moltiplicatore di Booth						0	−1	+1	−1	0	
	0	0	0	0	0	0	0	0	0	0	
	1	1	1	1	0	1	0	0	1		
	0	0	0	1	0	1	1	1			
	1	1	1	0	1	0	0	1			
	0	0	0	0	0	0	0				
Prodotto	1	1	0	1	1	1	0	1	1	0	

$-23 \times 6 = -138$

Moltiplicando					1	0	1	0	0	1	×
Moltiplicatore di Booth							+1	0	−1	0	
	0	0	0	0	0	0	0	0	0	0	
Compl. a 2 del Moltiplicando→	1	1	1	1	0	1	0	0	1		
	0	0	0	0	0	0	0	0			
	0	1	0	1	0	0	1				
Prodotto	1	1	0	1	1	1	0	1	1	0	

(Continuazione a pagina seguente.)

B.20 La ragione è quella di permettere l'immediato riconoscimento delle entità che non sono numeri normalizzati.

B.21 Quando $0 < esp < 255$ (in singola precisione) si ha a che fare con un numero normalizzato. Il fatto che il primo 1 sia quello nascosto, cioè a sinistra della virgola e non a destra come quando il bit nascosto è 0, fa guadagnare un bit nella rappresentazione della mantissa, a vantaggio della precisione.

$-23 \times -6 = 138$

Moltiplicando					1	0	1	0	0	1	$\times$
Moltiplicatore di Booth						0	-1	$+1$	-1	0	
	0	0	0	0	0	0	0	0	0	0	
	0	0	0	0	1	0	1	1	1		
	1	1	1	0	1	0	0	1			
	0	0	1	0	1	1	1				
	0	0	0	0	0	0					
Prodotto	0	0	1	0	0	0	1	0	1	0	

Tabella B.14 (Esercizio B.18) Differenti casi del prodotto col metodo di booth

B.22 Considerando quanto detto al Paragrafo B.7 si ha:
$14,3_{10} = 1110,01001100110011001100_2 = 1,11001001100110011001100 \times 10^{011}$,
Quindi la trasformazione richiesta è:
Segno=0;
Mantissa=11001001100110011001100
Esponente=10000010 (01111111+011)

B.24 Si può verificare che, se l'interpretazione è quella di numero intero, si tratta del numero 105, se l'interpretazione è quella testuale, si tratta del "i".

B.25 Ponendo la sequenza in ordine testuale e rappresentandola a byte si ha:

4C 31 6A 79 33 37

Dalla tabella di codifica ASCII si ottiene la stringa

L 1 y 3 7

B.26 Mostriamo la soluzione in forma del tutto generale. Si assume che la stringa di caratteri ASCII sia memorizzata nel vettore V di dimensioni pari alla lunghezza della stringa. Si assuma inoltre che tutti i caratteri contenuti in V siano effettivamente caratteri corrispondenti ai 10 digit decimali (0,1,...9). L'algoritmo di conversione della stringa in rappresentazione binaria può essere schematizzato come:

$$
\begin{array}{l}
Z \leftarrow 0 \\
\text{for } i = 1 \text{ to } n \\
Z \leftarrow Z \times 10 + num(V[i])
\end{array}
$$

dove *num(c)* è la funzione che trasforma la codifica ASCII di un carattere numerico nel corrispondente valore. In pratica si tratta di sottrarre dalla codifica ASCII del carattere il numero esadecimale 30 corrispondente alla codifica di "0". Nel caso specifico la stringa "2836" è codificata in memoria con questa sequenza esadecimale: 32 38 33 36.

B.27 Se si effettua la somma tra la rappresentazione binaria di due digit BCD si hanno tre possibili risultati:

- la somma è minore di 9: il risultato è già codificato in BCD (es.: $3 + 2 = 5$);

- la somma eccede 9 ma non dà riporto: occorre ricodificare il risultato in BCD aggiungendo 6. Da questa situazione può risultare un riporto per la cifra di sinistra (es.: $9 + 2 = 11$ ovvero $1001 + 0010 = 1011 \rightarrow 1011 + 0110 = 1\,0001$);
- la somma eccede 9 e dà riporto (es.: $9+8 = 17$); anche in questo caso occorre ricodificare il risultato aggiungendo 6.

Numero	Codifica BCD
79	0111 1001
48	0100 1000

Tabella B.15 (Esercizio B.27) Codifica BCD dei numeri 79 2 48: i singoli raggruppamenti di quattro bit del codice BCD costituiscono la codifica binaria delle cifre decimali.

L'esecuzione della somma procede da destra verso sinistra. Nel caso specifico si ha:

- $9 + 8 = 17$ ovvero $1001 + 1000 = 10001$ che ricodificato in BCD (sommando 6) è 0001 0111. Dunque la cifra a destra della somma è 7 con riporto 1.
- $7 + 4 + 1 = 12$ ovvero $0111 + 0100 + 0001 = 1100$ che ricodificato in BCD (sommando 6) è 0001 0010. Dunque la somma di 7 con 4 e col riporto della cifra a destra è 2 con riporto di 1. In conclusione il risultato è 127.

In Tabella B.16 viene mostrato il processo relativo a questa somma. In Figura B.25 lo schema a blocchi dell'algoritmo di somma di due cifre BCD.

	0	1	1	1		1	0	0	1	79
	0	1	0	0		1	0	0	0	48
	1	1	0	0	1	0	0	0	1	(il riporto su questa riga si somma a sinistra)
	0	1	1	0		0	1	1	0	somma 6
1	0	0	1	0		0	1	1	1	
1		**2**					**7**			

Tabella B.16 (Esercizio B.27) Somma BCD di 79 con 48 attraverso una somma binaria e normalizzazione.

B.28 Le reti richieste sono in Figura B.28. Mostriamo come si perviene a quella di NOR.

$$
\begin{aligned}
R &= AB = \overline{\overline{AB}} = \overline{\overline{A}+\overline{B}} = \overline{A} \downarrow \overline{B} \\
S &= A\overline{B} + \overline{A}B = \overline{\overline{A\overline{B}+\overline{A}B}} = \overline{\overline{(A\overline{B})}\,\overline{(\overline{A}B)}} = \overline{(\overline{A}+B)(A+\overline{B})} = \\
&= \overline{\overline{A}A + \overline{A}\,\overline{B} + BA + B\overline{B}} = \overline{\overline{A}\,\overline{B}+AB} = \overline{\overline{A+B} + \overline{\overline{A}+\overline{B}}} = (A\downarrow B)\downarrow(\overline{A}\downarrow\overline{B})
\end{aligned}
$$

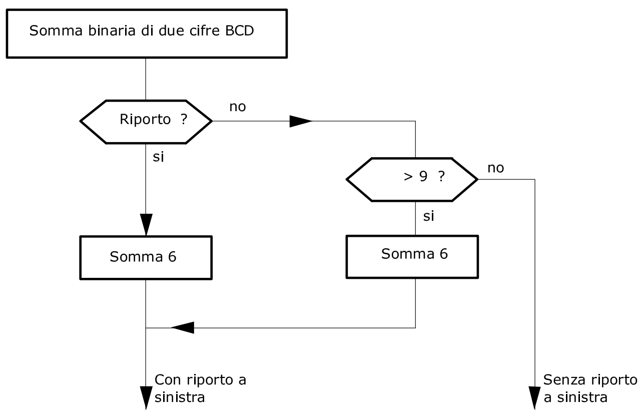


Figura B.25 (Esercizio B.27) Algoritmo per la somma di due cifre BCD attraverso un sommatore binario di 4 bit. Il blocco "somma 6" normalizza la rappresentazione binaria a rappresentazione BCD.

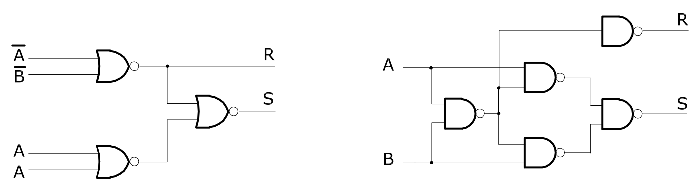


Figura B.26 (Esercizio B.28) Semisommatore con sole porte NOR e con sole porte NAND.

B.31 La differenza $A - B$ si effettua come somma di A con il complemento a 2 di B. Qui sotto i numeri sono rappresentati su 4 bit.

7	5	-7	-5
0111	0101	1001	1011

I casi $(A - B)$ sono i seguenti.

1. A e B positivi, con $A > B$. Specificatamente $7 - 5 = 2$; in binario $0111 - 0101 = 0010$. La differenza di due positivi ha dato luogo a un positivo. Risulta cioè verificata la condizione $\overline{A}_3\overline{B}_3\overline{O}_3 = 1$.
2. A e B positivi, con $A < B$. Specificatamente $5-7 = -2$; in binario $0101-0111 = 1\,1110$. La differenza di due positivi ha dato luogo a un negativo. Risulta verificata la condizione $\overline{A}_3\overline{B}_3O_3 = 1$.
3. A e B negativi, con $|A| > |B|$, ovvero $A < B$. Specificatamente $-7 - (-5) = -7 + 5 = -2$; in binario $1001 - 1011 = 1\,1110$. La differenza di due negativi ha dato luogo a un negativo. Risulta verificata la condizione $A_3B_3O_3 = 1$.
4. A e B negativi, con $|A| < |B|$, ovvero $A > B$. Specificatamente $-5-(-7) = -5+7 = 2$; in binario $1011-1001 = 0010$. La differenza di due negativi ha dato luogo a un positivo. Risulta verificata la condizione $A_3B_3\overline{O}_3 = 1$.

B.32 Riportiamo i passi del processo di esecuzione del prodotto -3×7 in Tabella B.17.

Passo	Operazione	R	P	M	X	Condizione ⇒ Conseguenza
0	Inizializzazione	1	0000	1101	0111	LSB=1 ⇒ LOAD
1	P ← P+M	1	1101	1101	0111	
	SHR(R||P||X)	1	1110	1101	1011	LSB=1 ⇒ LOAD
2	P ← P+M	1	1011	1101	1011	
	SHR(R||P||X)	1	1101	1101	1101	LSB=1 ⇒ LOAD
3	P ← P+M	1	1010	1101	1101	
	SHR(R||P||X)	1	1101	1101	0110	LSB=0 ⇒ $\overline{\text{LOAD}}$
4	–	1	1101	1101	0110	
	SHR(R||P||X)	1	1110	1101	1011	Risultato finale su P-X

Tabella B.17 (Esercizio B.32) Procedimento di esecuzione del prodotto di 1101 (-3) con 0111 (7) con la rete di Figura B.20. Al termine il risultato 1110 1011 si trova nella coppia di registri P - X. Si può verificare che si tratta di -21.

B.34 Consideriamo il caso della somma di due numeri positivi, ad esempio $A = 6$ e $B = 7$, che su 5 bit sono $A = 00110$ e $B = 00111$.
Ogni volta: (a) si sommano i due bit che si presentano con il riporto (inizialmente a 0); e (b) si scorrono a destra i numeri dati. Riportiamo qui di seguito la sequenza dei bit sommati e il risultato, secondo quest'ordine $A_i + B_i + R_{i-1} \rightarrow S_i, R_i$

(0) $0+1+0 \rightarrow 1,0$ (1) $1+1+0 \rightarrow 0,1$ (2) $1+1+1 \rightarrow 1,1$
(3) $0+0+1 \rightarrow 1,0$ (4) $0+0+0 \rightarrow 0,0$

Riordinando gli S_i, ovvero costruendo la sequenza $S_4S_3S_2S_1S_0$, si ottiene 01101.

Facciamo ora il caso di sommare un numero negativo ($A = -6$) e uno positivo ($B = 3$), ovvero di sommare due numeri di cui il primo negativo e il secondo positivo con $|A| > B$. In binario $A = 11010$ e $B = 00011$.
In questo caso la sequenza di somme è

(0) $0+1+0 \rightarrow 1,0$ (1) $1+1+0 \rightarrow 0,1$ (2) $0+0+1 \rightarrow 1,0$
(3) $1+0+0 \rightarrow 1,0$ (4) $1+0+0 \rightarrow 1,0$

Riordinando si ottiene 11101. Ovvero -3, come deve essere.
Si invita il lettore a verificare anche la somma di due numeri negativi.

Si noti che è del tutto irrilevante cosa viene immesso a sinistra in RA e RB, come è del tutto irrilevante il contenuto iniziale di RC.

Bibliografia

[Bar91] T. C. Bartee, *Computer architecture and logic design*, McGraw-Hill, 1991.

[HP06] J. L. Hennessy and D. A. Patterson, *Computer architecture: A quantitative approach*, Morgan Kaufman, 2006, Quarta edizione; esistono versioni in italiano.

[HVZ02] V. C. Hamacher, Z. G. Vranesic, and S. G. Zaky, *Computer organization*, McGraw-Hill, 2002, 5th edition.

[IEE85] IEEE, *Ieee standard for binary floating-point arithmetic*, Institute of Electrical and Electronics Engineers, ANSI/IEEE Standard 754-185, Agosto 1985.

[Omo94] A. R. Omondi, *Computer arithmetic systems. algorithms, architecture and implementation*, Prentice-Hall, New York, 1994.

[PH07] D. Patterson and J. Hennessy, *Computer organization and design - the hardware-software interface*, Morgan Kaufman, 2007, Terza edizione; esistono versioni in italiano.

[Rus70] B. Russell, *Introduzione alla filosofia matematica*, Newton Compton Italia, Roma, 1970.

Indice analitico

Indice